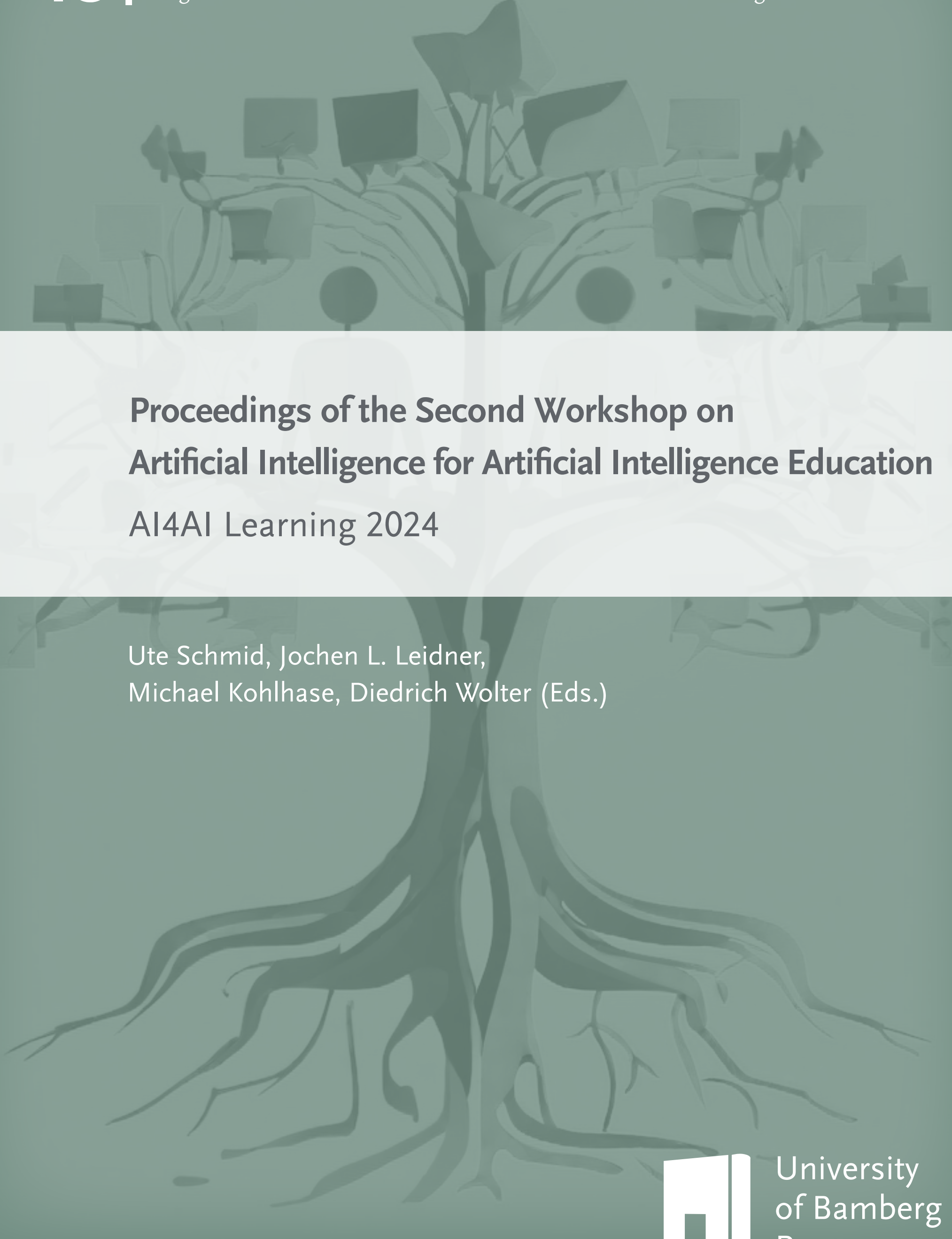
45 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg



Proceedings of the Second Workshop on Artificial Intelligence for Artificial Intelligence Education

AI4AI Learning 2024

Ute Schmid, Jochen L. Leidner,
Michael Kohlhase, Diedrich Wolter (Eds.)

University of Bamberg Press

45 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Contributions of the Faculty Information Systems and Applied Computer Sciences of the Otto-Friedrich-University Bamberg

Schriften aus der Fakultät Wirtschaftsinformatik
und Angewandte Informatik der Otto-Friedrich-
Universität Bamberg

Contributions of the Faculty Information Systems
and Applied Computer Sciences of the
Otto-Friedrich-University Bamberg

Band 45

University
of Bamberg
Press
2025

Proceedings of the Second Workshop on Artificial Intelligence for Artificial Intelligence Education

AI4AI Learning 2024

Ute Schmid, Jochen L. Leidner,
Michael Kohlhase, Diedrich Wolter (Eds.)

University
of Bamberg
Press
2025

Bibliografische Informationen der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Umschlaggestaltung: University of Bamberg Press
Umschlaggraphik: Erstellt mit „Stable Diffusion“

ORCID

Ute Schmid	https://orcid.org/0000-0002-1301-0326
Jochen L. Leidner	https://orcid.org/0000-0002-1219-4696
Michael Kohlhase	https://orcid.org/0000-0002-9859-6337
Diedrich Wolter	https://orcid.org/0000-0001-9185-0147

ISSN: 1867-6197 (Print) eISSN: 2750-8560 (Online)
eISBN: 978-3-98989-054-1 (Online)

URN: urn:nbn:de:bvb:473-irb-1076617
DOI: https://doi.org/10.20378/irb-107661

Content

Organisation

Workshop Organisers

Ute Schmid, University of Bamberg, Germany

Jochen L. Leidner, Coburg University of Applied Sciences, Germany and University of Sheffield, UK

Michael Kohlhase, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Diedrich Wolter, University of Lübeck, Germany

All workshop organizers are principal investigators in the joint project "Learning from Learners" (VoLL-KI), funded by the Federal Ministry of Education and Research (BMBF) as part of the Federal-State Initiative AI in Higher Education.

Program Committee

Preface

Education as an application domain of AI encompasses manifold important use cases and specialized settings, ranging from early education to advanced university programs. Currently, several European countries attribute funding to improving AI education and it thus appears most natural to employ AI techniques for this purpose as well. This workshop brings together researchers involved in these diverse European programs investigating, developing or exploring AI techniques for AI education.

The Second Workshop on Artificial Intelligence for Artificial Intelligence Education (AI4AILearning) has been held on September 24, 2024, in conjunction with the 47th German Conference on Artificial Intelligence in Würzburg, Germany. It follows the first workshop AI4AILearning which has been held in conjunction with ECAI 2023 in Krakau, Poland. The workshop provided a platform for exchange of ideas and experiences under the general theme of AI for education, specializing on university education in AI. We also wish to contribute towards forming a European community on the theme of AI for AI education and foster basic research as well as the development of intelligent assistance technology in a multidisciplinary setting in order to improve AI education by making use of AI technology itself: AI4AI.

In the workshop, six contributions were presented in full or short talks, along with an additional demonstration. The technical contributions are published in this collection. The contributions discussed at the workshop tackle different use cases of AI in teaching AI. One group of papers is concerned with the development of intelligent tutor systems for improving programming education that is foundational to most AI curricula. Another group of papers investigates the effect of employing AI techniques in education, especially considering machine learning techniques. The AI4AI workshop also featured discussions and a keynote by Adish Singla (Max Planck Institute for Software Systems, Germany). Organization of the workshop was supported by additional reviewers, whom the organizers gratefully acknowledge. Furthermore, we thank Adrian Völker for technical support of the workshop and this proceedings volume.

The organizers hope that the intense discussions at the workshop will continue and contribute to improving the ability of the AI community to educate the next generation of AI research.

September, 2024

Ute Schmid
Jochen L. Leidner
Michael Kohlhase
Diedrich Wolter

Invited Talk: AI-driven Educational Technology for Computational Thinking and Programming

Adish Singla

Machine Teaching Group, Max Planck Institute for Software Systems, Saarbrücken, Germany
adishs@mpi-sws.org

Abstract

Computational thinking and basic programming skills are essential for everyone in the 21st century, both for students and adults, to thrive in the digital society. Consequently, there is an increasing emphasis on introducing computing and programming education at an early age, starting at elementary-level grades. However, given the conceptual and open-ended nature of programming tasks, novice learners often struggle when solving programming tasks by themselves. Given the scarcity of human tutoring resources to provide individualized assistance, AI-driven educational technology has the potential to provide scalable and automated assistance to struggling learners. In this talk, I will present our work on AI-driven programming education empowered by automated techniques for synthesizing new practice tasks, generating personalized feedback, and modeling learners' knowledge. I will describe unique challenges and opportunities in the programming domain, which can also drive the next scientific breakthroughs in AI-driven education for other subject domains. I will conclude with a discussion of crucial ingredients to succeed in making programming education easier and accessible for all.

Chat with your Lecture Recording – Creating easy-to-use Chatbots for Learning Videos

Sebastian Hobert

TH Lübeck

sebastian.hobert@th-luebeck.de

Abstract

Educational chatbots represent a promising technology for improving learning. Prior research studies revealed the benefits of using chatbots in educational settings, e.g., to enable students to engage with learning content. However, the technology is not widely used in teaching practice. In response to this gap between scientific findings and usage of the technology in practice, we aim to enable lecturers to create their own customized chatbots based on educational videos by developing a chatbot creation tool that can create tailored chatbots (semi-)automatically. Following a design science research approach, we iteratively designed a software prototype that can convert lecture recordings or similar learning videos into chatbots. The specialty of the prototype is that it is designed to make the chatbot technology accessible to lecturers by reducing technical barriers. With the help of the prototype, it is possible to adapt the chatbot to the lecturer's teaching content without the need for a time-consuming, manual creation of a knowledge base. Our evaluation with experienced lecturers, instructional designers, and developers of educational technology indicate positive results. In this paper, we present the results of our development and evaluation process.

Keywords Chatbot, Technology-Enhanced-Learning, Video-based Learning, Large Language Model, Education

Introduction

Chatbots have been used and researched in educational settings for years (see, e.g., Hobert & Meyer von Wolff, 2019; Winkler & Söllner, 2018; Wollny et al., 2021). Many different use cases have been analyzed in the past. Use cases for chatbots in educational settings are, for instance, supporting students in language learning (Huang et al., 2022), supporting students in learning programming (Hobert, 2023), or improving video-based teaching by integrating chat interactions (Winkler et al., 2020).

When integrating chatbots into online learning settings, the engagement of users with the learning content typically changes. Typical passive interactions with learning content like reading textual learning modules or watching videos can be extended by conversations with a chatbot. This has been found to be supporting for the learning process, e.g., for engagement, learning, and self-efficacy (Chang et al., 2022).

Although positive effects of well-designed chatbots on the learning process are to be expected, and many use cases have already been researched, no widespread use of chatbots to support learning can be identified in teaching practice. Grounded in our experience of integrating educational chatbots into multiple field settings in the past (e.g., Hobert, 2023; Hobert & Berens, 2024), we consider various reasons to be particularly important: (1) Teachers need easy-to-use tools to create their own chatbots. (2) Creating sufficient materials for training a chatbot manually is time-exhaustive. (3) Available chatbot tools are often aimed at technology-affine users.

The emergence of large language models (LLMs) seems to be a catalyst for educational chatbots. Available tools make it easy even for less technology-affine users to create customized chatbots for various settings. However, a prerequisite for this is the availability of textual content (such as slides, books, or scripts) specifically aligned to the course. In teaching practice in formal learning scenarios, however, it can be experienced that although the available learning materials (often slides) provide a basic building block, much additional knowledge is conveyed orally in face-to-face lectures, video conferences, or on the spoken track of recorded learning videos. If teachers are required to reproduce all of the knowledge conveyed orally in full in writing, this, in turn, involves a great deal of additional effort.

Based on this problem and with the aim of easily integrating chatbots into our own AI courses, we aim to make chatbots easy to create and easy to customize based on lecture recordings. To this end, we follow a design science research approach (Hevner, 2007; Hevner et al., 2004) to design and evaluate an educational technology tool that allows lecturers to create customized chatbots based on learning videos. The underlying idea is that lecturers do not have to spend any additional time creating written teaching materials but can simply use a recording of their lecture as a basis. Thus, we aim to answer the following research question: How to enable lecturers to create their own customized chatbots based on educational videos? To answer this research question, we build on our experiences from various previous chatbot studies. In contrast to these prior studies (e.g., Hobert, 2019, 2023; Hobert et al., 2023; Hobert & Berens, 2024; Winkler et al., 2020), in which we spent a lot of time and effort to manually create knowledge bases, the further development in this project is to automate the creation of a knowledge base using the technology of LLMs. In doing so, we want to take a small but relevant step towards the further dissemination of chatbots in educational settings.

Theoretical Background

Educational chatbots can be defined as software programs that interact with learners using natural language (Følstad & Brandtzaeg, 2020; Hobert & Meyer von Wolff, 2019) in educational settings (like lectures or online courses). By providing a chat-based user interface, educational chatbots aim to mimic human-like interactions (Følstad & Brandtzaeg, 2020; Weizenbaum, 1966), e.g., about learning content or organizational aspects of learning.

In recent years, educational chatbots received substantial research interest, which is reflected in the emergence of several literature reviews (e.g., Hobert & Meyer von Wolff, 2019; Winkler & Söllner, 2018; Wollny et al., 2021). In prior research studies, many purposes and use cases have been researched (like language learning (Huang et al., 2022), programming (Hobert, 2023), enhancing video-based learning (Winkler et al., 2020), or corporate training (Hobert et al., 2023)). Summarizing the results of prior research shortly, it can be concluded that educational chatbots seem to be able to foster students' learning in many different use cases.

From a theoretical point of view, the benefit of including chatbots in educational settings for supporting learning processes can, for instance, be explained with the ICAP framework (Chi & Wylie, 2014). According to the ICAP framework, it is to be expected that learning success can be fostered by increasing learning engagement (Chi & Wylie, 2014). Purely text-based or video-based online learning formats can often be regarded as rather passive. This can also apply to (large) lecture formats with limited interaction. According to the framework, however, interactive formats, such as discussions, are considered to be beneficial for successful learning processes. This is where educational chatbots can support the learning process, as they can discuss and interact with users (even in purely online formats) using natural language. Therefore, it is to be expected that more interactive learning settings can be created with the help of educational chatbots.

Building on the promising prior research results and the theoretical grounding in the ICAP framework (Chi & Wylie, 2014), this research project aims to support lecturers in integrating educational chatbots in their educational settings to improve students' learning.

Research Design

In this project, we follow a design science research-oriented research design. We specifically base our methodology on the three-cycle design science research framework (Hevner, 2007; Hevner et al., 2004). To develop our software prototype and the underlying technical process to create chatbots (semi-)automatically from learning videos, we first ground our design process on our own prior research studies on chatbots in the educational sector (e.g., Hobert, 2019, 2023; Hobert et al., 2023; Hobert &

Berens, 2024; Winkler et al., 2020) and related learning theories (i.e., the ICAP framework (Chi & Wylie, 2014)). Additionally, we ground our research results on the application context (i.e., the environment) and the expertise of experienced educators.

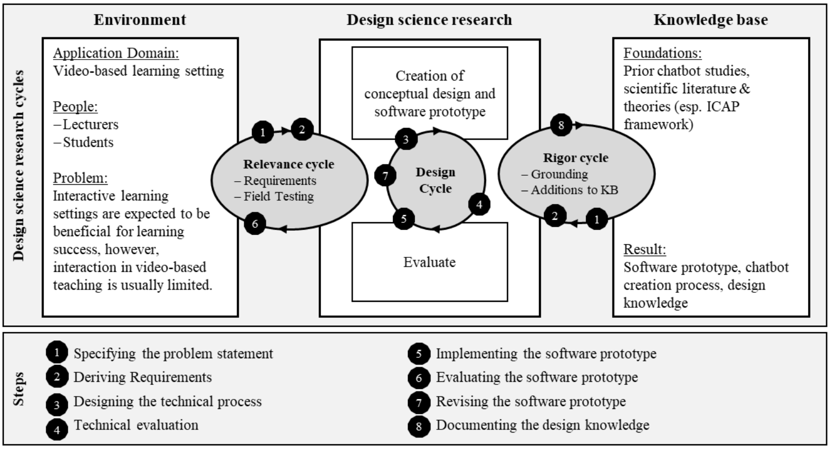

Design science research cycles
Environment
Application Domain:
Video-based learning setting
People:
–Lecturers
–Students
Problem:
Interactive learning settings are expected to be beneficial for learning success, however, interaction in video-based teaching is usually limited.
Design science research
Creation of conceptual design and software prototype
Evaluate
Relevance cycle
– Requirements
– Field Testing
Design Cycle
Rigor cycle
– Grounding
– Additions to KB
Knowledge base
Foundations:
Prior chatbot studies, scientific literature & theories (esp. ICAP framework)
Result:
Software prototype, chatbot creation process, design knowledge
Steps
1 Specifying the problem statement
2 Deriving Requirements
3 Designing the technical process
4 Technical evaluation
5 Implementing the software prototype
6 Evaluating the software prototype
7 Revising the software prototype
8 Documenting the design knowledge

Fig. 1. Design science research process (adapted from (Hevner, 2007; Hevner et al., 2004)).

As displayed in **Fig. 1**, the research process consists of eight consecutive steps. First, we derive the underlying research problem from the educational practice (relevance cycle) as well as from prior research studies and learning theory (rigor cycle). Based on this, we derive software requirements and design the technical process to (semi-)automatically create a chatbot from an educational video. In the fourth step, we technically evaluate the capabilities of the technical process before we implement a web-based software prototype in the fifth step. In an evaluation study with experienced lecturers, instructional designers, and developers of educational technology, we evaluate the software prototype to identify suggestions for improvements. Finally, we revise the prototype and document the design knowledge in this paper.

Design and Evaluation

Specifying the Problem Statement

As motivated in the introduction section, we aim to improve the interaction of students with learning content by enabling lecturers to create educational chatbots easily. In typical video-based learning settings, students watch learning videos (e.g., lecture recordings, micro learning videos, or recordings of hands-on tutorials) in a rather passive way. Interacting with learning materials in a passive way is, however, expected to be less successful in terms of fostering learning success compared to active, constructive, or even interactive learning settings, according to the ICAP framework (Chi & Wylie, 2014).

Fostering the interactivity in asynchronous video-based learning is, however, not easy as students typically watch videos not in a formal learning environment (e.g., at home for the exam preparation). One possibility to increase the interactivity is the integration of chatbots (i.e., pedagogical conversational agents). For instance, in a prior research study it could be shown that learning success in a video-based learning setting can be fostered when specifically designed chatbots are integrated to interact with students about the video's learning content (Winkler et al., 2020).

Integrating chatbots in a similar way as it was done in Winkler et al., (2020) is, however, challenging for lecturers as it requires a lot of additional (manual) work. This can particularly be explained by the required design process of a tailored chatbot. So far, as in Winkler et al., (2020), the learning materials must be specifically prepared to be accessible by a chatbot. In a learning video and specifically in a lecture recording, the learning contents are typically not fully available in written form and can, thus, not easily be integrated into common chatbot platforms. To address this problem, we aim to extract not only the learning content from lecture slides but also the lecturer's voice-based instructions from learning videos. Using this extracted information, we aim to automatically create a knowledge base that can be used by an LLM-based chatbot to answer students' questions. In doing so, we are focusing on a content level in which a chatbot is provided that can answer content-related questions about the given learning video (e.g. "What is the definition of term XY?", "Please explain term XY again in different words", "Please summarize the most important aspects of the video.").

Deriving Requirements

As outlined in the introduction section and based on the problem statement, the software to be designed should be able to create chatbots in an easy-to-use way even by less technology-affine lecturers based on educational videos. To this end, we derived requirements as the basis for the subsequent implementation process (see **Table 1**). First, the software to be designed needs suited functionalities to select and upload a learning video (R1). Ideally, the lecturer should not need to care about specific video codecs or formats as this would hinder less technology-affine lecturers. After uploading the video, a (semi-)automated process should start to create the chatbot's knowledge base and initiate the chatbot. To this end, the learning content of the uploaded video needs to be extracted (R2). This should include the learning content from slides as well as the lecturer's voice-based instructions. Based on the extracted learning content, a suitable knowledge base should be created automatically (R3) that can be used by state-of-the-art chatbot technology (i.e., LLMs) as an input to provide chatting functionalities using an easy-to-use user interface for students (R4). The whole creation process should be provided in an easy-to-use user interface guiding lecturers step-by-step through the creation process without needing specific technical knowledge (R5).

Table 1. Overview of the core requirements for the software prototype.

#	**Description**
R1	Provide functionalities to select and upload arbitrary learning videos
R2	Provide functionalities to automatically extract the learning content from a learning video (including the content from slides and the lecturer's voice)
R3	Provision of functionalities for the automatic creation of a suitable knowledge base based on the automatically extracted learning content
R4	Provide chatting functionalities using an easy-to-use user interface for students
R5	Provide an easy-to-use user interface guiding lecturers through the creation process without needing specific technical knowledge

Designing the Technical Process

Based on the derived problem statement and the corresponding requirements, we designed the technical process visualized in **Fig. 2**. We divided the chatbot creation process into four subprocesses: (1) video pre-processing, (2) extraction, (3) LLM-based pre-processing, and (4) chat.

At the beginning of the video pre-processing, the video must first be uploaded. After completing the upload, the video can be analyzed to detect the different slides. To do this, the video stream is analyzed and it is detected when the visual content changes substantially. This is interpreted as a slide transition and is used for splitting the video into chunks. After splitting the video into smaller chunks (aligned with the slides), the first subprocess terminates. In the second subprocess, the learning content is extracted from the video chunks. For each video chunk, the textual content of the slide is extracted using optical character recognition (OCR). Additionally, the audio of the video chunk is extracted and transcribed using speech-to-text (STT).

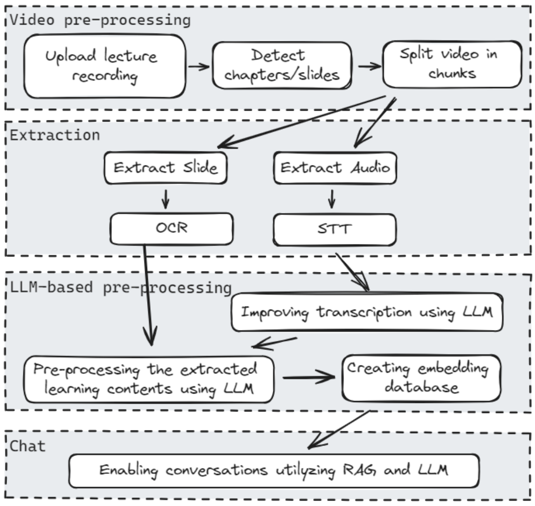

Video pre-processing
Upload lecture recording
Detect chapters/slides
Split video in chunks
Extraction
Extract Slide
Extract Audio
OCR
STT
LLM-based pre-processing
Improving transcription using LLM
Pre-processing the extracted learning contents using LLM
Creating embedding database
Chat
Enabling conversations utilyzing RAG and LLM

Fig. 2. Derived process to automatically create a chatbot from educational videos

Using the extracted textual information from OCR and STT as the input, the LLM-based pre-processing starts. First, using an LLM the STT-based textual extraction of the speaker's voice is improved. To this end, the LLM is prompted to "correct only spelling mistakes and grammatical errors". This step seems particularly helpful as automatic transcriptions using STT are – according to our experience – often prone to grammar and spelling issues. The reason for this is that lecture recordings are often recorded spontaneously in the lecture hall without following a detailed manuscript. The spoken language, therefore, often contains mistakes that do not typically occur in written learning materials. Using an LLM for improving automatic transcriptions works well – at least for our tests during this research project. The improved transcriptions, as well as the extracted texts from the slides (OCR), are then further processed. Creating summaries of all slides and the entire lecture video and extracting key points seem to be particularly helpful. To this end, we prompt the LLM to create such summaries for the extracted content. For instance, we prompt the LLM to "summarize the text in one sentence" or to "list the most important key points”. These summaries and key points are then used to create an embedding database together with the extracted and improved content of the OCR and STT steps.

Finally, a chatbot can be provided using retrieval augmented generation (see, e.g., aws.amazon.com, 2024). With this method, the chatbot is then able to respond to the user’s questions. To this end, the user first need to ask the chatbot a question. This question is then passed to the embeddings database to conduct a similarity search. The most relevant results from the similarity search are then passed to the LLM, which is promoted to “answer the question briefly in two sentences” and to “use the following information from [the similarity search]” as a basis. This has the effect that the chatbot’s response depends to a large extent on the results of the similarity search and the answer generation by the LLM. Since the processed content of the learning video is stored in the embedding database (see previous steps), the generation of the response is aligned to the content of the learning video.

Based on the technical design of the process, we implemented it as a collection of multiple small Python and bash scripts using common open-source tools and libraries (like pytesseract (Github, 2024c) for OCR, whisper (Github, 2024a) for STT, chroma (trychroma.com, 2024) for the embedding database, and ffmpeg (ffmpeg.org, 2024) for analyzing and splitting the video). This collection of scripts was used as the input for the technical evaluation and allowed the designed technical process to be fully tested.

Technical Evaluation

Before implementing a full user interface to execute the designed technical process, we aimed to test the technical process manually in a first technical evaluation. To this end, we selected ten different learning videos from our own lectures and ran the full process manually. Afterward, we manually tested the quality of chat conversations.

To generate exemplary chat messages, we use exemplary user messages retrieved from prior chatbot studies that we conducted in previous years (e.g., (Hobert, 2023; Hobert & Berens, 2024)). For instance, we tested the following messages: *Please define the term $topic. What is meant by $topic? What is the video about? On which slide can I find more information about $topic?*

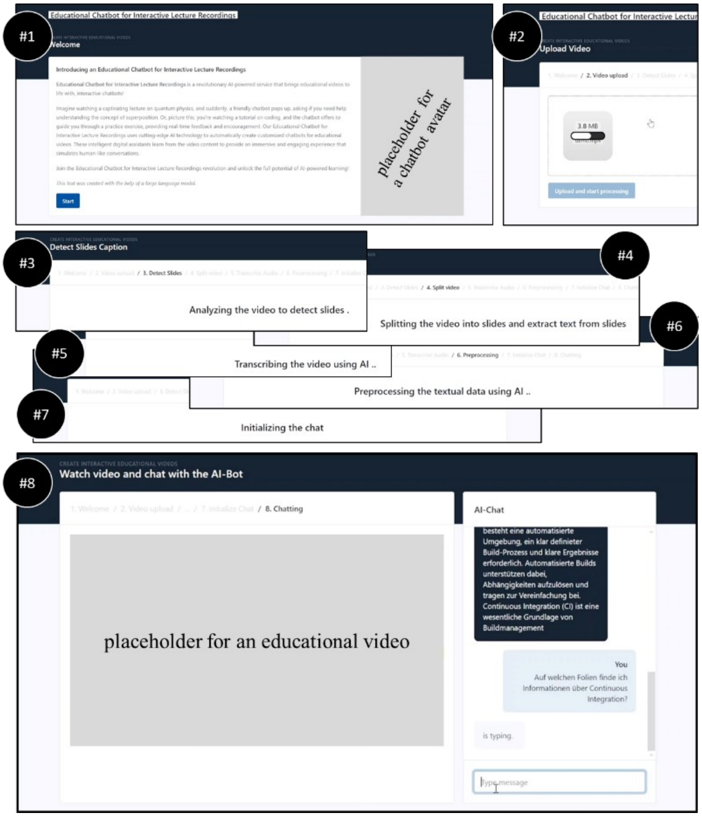
Whereas questions about getting a summary or about defining specific terms mentioned in the videos resulted in good-quality answers, getting information about where to find specific information (e.g., on which slide) was less successful. The reason for this was that we did not provide the slide number as an information to the LLM. After adding this information to the database, the answer quality increased, and the LLM was capable of responding to these types of questions as well. After getting a sufficiently high answer quality with our ten exemplary learning videos, we decided to continue with the implementation of a fully functional software prototype.

Implementing the Software Prototype

To enable lecturers – even without specific technical knowledge – to create chatbots for their lecture recordings easily, we implemented a web-based software prototype using Python's Flask framework (Github, 2024b). To design an easy-to-use user interface, we rely on the MIT-licensed Tabler UI kit (The Tabler Authors, 2024).

To implement the technical process, we refined our script collection implemented during the design of the technical process and integrated it into our web-based user interface.

As visualized in **Fig. 3**, the resulting user interface guides the lecturers step-by-step through the creation process. The procedure consists of one introductory welcome screen (step #1), six intermediate steps (#2 – #7) for video pre-processing, extraction, and LLM-based pre-processing, and finally, the user may directly interact with the resulting chatbot while being able to watch the video at the same time (#8).


#1
Educational Chatbot for Interactive Lecture Recordings
Welcome
Start
placeholder for a chatbot avatar
#2
Upload Video
2. Video upload
3.8 MB
Upload and start processing
#3
Detect Slides Caption
3. Detect Slides
#4
4. Split video
#5
#6
6. Preprocessing
#7
#8
Watch video and chat with the AI-Bot
8. Chatting
placeholder for an educational video
AI-Chat
You
is typing.
Type message

Introducing an Educational Chatbot for Interactive Lecture Recordings

Educational Chatbot for Interactive Lecture Recordings is a revolutionary AI-powered service that brings educational videos to life with, interactive chatbots!

Imagine watching a captivating lecture on quantum physics, and suddenly, a friendly chatbot pops up, asking if you need help understanding the concept of superposition. Or, picture this: you're watching a tutorial on coding, and the chatbot offers to guide you through a practice exercise, providing real-time feedback and encouragement. Our Educational Chatbot for Interactive Lecture Recordings uses cutting-edge AI technology to automatically create customized chatbots for educational videos. These intelligent digital assistants learn from the video content to provide an immersive and engaging experience that simulates human-like conversations.

Join the Educational Chatbot for Interactive Lecture Recordings revolution and unlock the full potential of AI-powered learning!

This text was created with the help of a large language model.

Analyzing the video to detect slides .

Splitting the video into slides and extract text from slides

Transcribing the video using AI ..

Preprocessing the textual data using AI ..

Initializing the chat

besteht eine automatisierte Umgebung, ein klar definieter Build-Prozess und klare Ergebnisse erforderlich. Automatisierte Builds unterstützen dabei, Abhängigkeiten aufzulösen und tragen zur Vereinfachung bei. Continuous Integration (CI) ist eine wesentliche Grundlage von Buildmanagement

Auf welchen Folien finde ich Informationen über Continuous Integration?

Fig. 3. Overview of the implemented software prototype

Evaluating the Software Prototype

To evaluate the software prototype, we created an online survey and distributed it among experienced lecturers, instructional designers, and developers of educational technology. Hereby, we chose a mixed-method evaluation, as similar approaches have been shown to be suited for evaluating and developing chatbot-based system in prior studies (e.g., Hobert, 2019; Hobert & Berens, 2024).

Within the online evaluation, the participants first received a written introduction to the overall context of the research project, including further information on the scientific background. Afterward, we asked the participants to self-evaluate their AI literacy using the Meta AI Literacy Scale (Carolus et al., 2023). We included the measurement of AI literacy to ensure that the participants had sufficient prior knowledge of AI. In the next step, the usage scenario was introduced in an executive summary and the software prototype with an exemplary learning video was demonstrated in a video screencast. Demonstrating the prototype in a screencast ensured that all participants had the chance to get insights into all steps of the prototype. After this demonstration step, the participants were asked to evaluate the software prototype using the user experience questionnaire (UEQ) scale (Laugwitz et al., 2008). Finally, we asked the participants for qualitative feedback to give us additional insights on how to improve the prototype. Hereby, we were particularly interested in aspects of the prototype that were considered to be useful and valuable, as well as possible suggestions for improvement.

Quantitative Feedback on the User Experience

The participants evaluated the user experience of the developed software prototype positively in all six sub-scales of the UEQ (Laugwitz et al., 2008). According to the benchmark included in the UEQ Data Analysis Tools (The UEQ Team, 2024), the software prototype was rated as excellent in terms of attractiveness, perspicuity, efficiency, simulation, and novelty. Only in the scale dependability, it receives lower but still positive ratings. This can maybe be explained by the nature of the creation process. During the semi-automated process, most steps are automatically executed by the system, and the results are dependent on the results of the large language model. This can maybe result in a feeling that the next steps are less predictable compared to traditional, not AI-supported, systems. Overall, the results of the user experience evaluation do not indicate any major problems.

Qualitative Feedback

To analyze the qualitative feedback collected as part of the evaluation procedure, we aggregated and categorized the mentioned aspects. We translated selected quotes into English to include them in this paper.

1. *Positive Aspects*

First, we analyzed the aspects of the prototype that were particularly named as useful and valuable. Related to this question, the participants named 33 aspects, which we aggregated. Many aspects were mentioned by several participants, so a total of four major aspects emerged after we finished our qualitative analysis.

The simplicity of the usage of the creation process of the chatbot was named by many of the participants. For instance, the participants highlighted the simplicity of the "automatic creation of the chatbot" and the "simple usage and process".

Closely related to simplicity is the intuitive user interface. Hereby, the participants stated that "the individual steps are displayed clearly", and "the look and feel correspond to the familiar interface of chats". The qualitative statements related to the user interface are in line with the quantitative feedback on the user experience.

The possibility to interact with the chatbot about the specific learning content of the uploaded video was named by multiple participants as a major benefit. For instance, one participant highlighted "the ability to interact with the chatbot, e.g., to give instructions and ask questions". The natural language conversation with the chatbot ("interaction using free text in the user's natural language") was also highlighted as a benefit.

Finally, the (semi-)automatic processing of the video and its content was evaluated as particularly valuable. The participants mentioned this aspect the most. For instance, participants highlighted that "the slides are automatically recognized from the video and [that] the text is automatically extracted", "the chatbot can specify the slides on which the answer is based on when asked", "the ability to extract and analyze both written and spoken language". These quotes highlight once again the usefulness of the implemented process to (semi-)automatically generate the chatbot based on the video.

2. *Suggestions for improvement*

The participants made 39 suggestions for improvement. Many of them were named multiple times. After aggregating them, we resulted in four aspects related to the creation process of the chatbot and two aspects related to the learners' interaction with the chatbot.

First, multiple participants requested the possibility to enable uploads for further documents in addition to the learning video. For instance, "additional textual lecture materials or literature sources" could be uploaded. One participant also suggested including the software artifact directly into a learning management system and enabling the import of the content of the entire course into the video.

In addition, multiple participants asked whether it would be possible to provide a multilingual chatbot that is capable to converse, for instance, in either German or English. This form of internationalization would enable non-native speakers to interact more easily with the learning content.

To make the creation process of the chatbot more transparent, it was suggested to provide "more detailed explanations in the intermediate steps" while the chatbot is created. This suggestion is somehow contradicting to the suggestion to "automate the next buttons for the intermediate steps", which was state by another participant.

One participant requested to offer better possibilities for quality assurance. For instance, she or he suggested offering the possibility to read the automatic transcription from the video. This seems to be an interesting suggestion at first glance, but it is – according to our opinion –

questionable whether this really would improve the quality of the output as the transcription are only one building block for the generation of the chatbot's answer using the LLM.

In addition to these suggestions for improvement targeting the creation process, two further aspects were suggested related to the learners' interaction with the chatbot.

First, it was requested by several participants to include the possibility to jump directly to the slide when the chatbot mentioned a specific slide: "It would be good if I [...] can jump directly to the point in the video where the information is mentioned". This would enable learners to watch the specific section of the video once again.

Finally, some participants asked to proactively offer learners the possibility to get a summary or the key message of the video. This could for instance be included in the starting point of the conversation.

According to our assessment, the suggestions for improvement seem to be helpful and might contribute to further improve our developed software artifact.

Nevertheless, we also want to disclose one mostly critical aspect mentioned by one participant who responded that "I can also just take the slides and search for terms using the search function. That would be more efficient on the one hand and would also show me 1:1 what is relevant without additional information." This statement seems controversial but misses – according to our estimation of the benefits of chatbots in educational settings – that chatbots allow learners a different way of interacting with the learning content using natural language conversations.

Revising the Software Prototype

Based on the suggestions for improvements revealed in the qualitative feedback, we revised our software implementation. In particular, we revised the prototype in four areas. (1) We enabled the possibility to upload not only one video during the creation process but users are now allowed to also upload further text documents at the same time. These text documents are processed similarly to the video without executing the transcription and OCR steps. This enables lecturers to integrate additional materials like handouts or speaker notes. (2) We added more explanatory information on the processing steps and also automated the next buttons at the same time (by moving to the next step after the processing steps has finished either when the user clicks the next button or after a timer of 10 seconds expired). (3) For learners, we revised the initial welcome message of the chatbot. We integrated a button to switch to English in the conversation. Additionally, the chatbot proactively offers the possibility to show a short summary of the learning video. (4) We added links to the chatbot's answer every time a specific slide number is referenced. By clicking on the slide number, the video jumps directly to the corresponding slide, and the user may replay the video again at this point.

Overall, we believe that the implemented suggestions for improvement make sense and that the revision improved the prototype. We believe that the implemented improvements do not require another evaluation with lecturers at this time. Instead, we suggest that the next step would be to test automatically generated chatbots in real learning settings with learners. This is subject to future research. At this point in time, we are ending this development process as part of our research project at this point.

Discussion and Conclusion

In this design science research project, we aimed to enable lecturers to create their own customized chatbots based on educational videos. To achieve this, we derived requirements, designed the underlying technical process, and developed a software prototype in multiple design cycles. The software prototype was evaluated by experienced lecturers, instructional designers, and developers of educational technology. Both the quantitative and qualitative evaluation results were encouraging and predominantly positive, so we assume that we have developed a helpful first prototype. Nevertheless, this paper presents only an intermediate step and educational chatbots should continue to be the subject of future research.

Implications for Spreading Educational Chatbots

Although research has targeted educational chatbots for many years and shows their benefits, we observe that they are still used rarely. With this research and development project, we aim to address this issue by enabling lecturers to create chatbots for their teaching easily. We see particularly two main use cases: (1) Lecturers in in-class teaching settings can record their lectures and use those lecture recordings to create tailored chatbots for their students. Students can then use the provided chatbot for their follow-up work and for the preparation for the final exam. By giving them an additional interactive way to engage with the learning content, to ask content-related questions, and to discuss relevant aspects of the lecture, a positive impact on the learning process is to be expected according to the ICAP framework (Chi & Wylie, 2014). (2) The developed software prototype can be used to improve online-based courses (like MOOCs or online study programs). These courses are, in many cases, pre-dominated by a substantial part of passive learning when watching videos. When integrating the ability to discourse with a chatbot about the learning content, it is to be expected that this could have a positive impact on the students' learning.

We assume that both use cases can be implemented with our developed software prototype by lecturers - even without specific technical expertise. We hope - also on the basis of the positive feedback from the evaluation - that we can contribute to the spread of the chatbot technology in the educational sector.

Implications for our AI Education

In addition to the implications for spreading the educational chatbot technology, we plan to integrated the developed software prototype as a show case of an AI use case in two AI courses. To this end, chatbots for supporting in-class teaching in both courses should be provided to the students. This enables them to interact with a tailored LLM-based chatbot and gives them the opportunity to interact with an LLM to analyze its possibilities and limitations. Afterward, the underlying architectures (e.g., LLM, RAG, STT) should be discussed.

Limitations and Future Research

The software developed as part of this project is currently a software prototype. Bugs known to us have been fixed as part of the iterative development process, but the prototype should be tested in more detail before going into production. For instance, it would be interesting to test in more detail how accurate the individual process steps (e.g., STT, TTS, or LLM-based improvements of the transcriptions) work. In addition to that, an evaluation with students is still pending. Here it would be particularly interesting to analyze how accurate the chatbot's answers are. As part of this step, it could also be further investigated how the system should deal with incorrect answers. For example, a retro perspective quality control, or a way for students to report answers that are then reviewed by a lecturer, could be feasible here. As a follow up study, we are interested in testing the chatbot in a field setting. We expect that this will provide us with further insights on how to improve the student-centered chat interface. By conducting a subsequent discourse analysis, we might be able to get additional insights into the students' interaction with the chatbot and it's answer quality. Overall, we also hope to gain further insights into whether the automatic creation of the knowledge base using our derived technical process can lead to a change in the discourse quality. As a comparison, we could rely on insights that we gained from several prior chatbot studies (e.g., Hobert, 2019, 2023; Hobert et al., 2023; Hobert & Berens, 2024; Winkler et al., 2020) in which knowledge bases were created manually.

References

aws.amazon.com. (2024). *What is RAG? - Retrieval-Augmented Generation AI Explained - AWS*. Amazon Web Services, Inc. https://aws.amazon.com/what-is/retrieval-augmented-generation/

Carolus, A., Koch, M. J., Straka, S., Latoschik, M. E., & Wienrich, C. (2023). MAILS - Meta AI literacy scale: Development and testing of an AI literacy questionnaire based on well-founded competency models and psychological change- and meta-competencies. *Computers in Human Behavior: Artificial Humans*, *1*(2), 100014. https://doi.org/10.1016/j.chbah.2023.100014

Chang, C., Hwang, G., & Gau, M. (2022). Promoting students' learning achievement and self-efficacy: A mobile chatbot approach for nursing training. *British Journal of Educational Technology, 53*(1), 171–188. https://doi.org/10.1111/bjet.13158

Chi, M. T. H., & Wylie, R. (2014). The ICAP Framework: Linking Cognitive Engagement to Active Learning Outcomes. *Educational Psychologist, 49*(4), 219–243. https://doi.org/10.1080/00461520.2014.965823

ffmpeg.org. (2024). *FFmpeg*. https://www.ffmpeg.org/

Følstad, A., & Brandtzaeg, P. B. (2020). Users' experiences with chatbots: Findings from a questionnaire study. *Quality and User Experience, 5*(1). https://doi.org/10.1007/s41233-020-00033-2

Github. (2024a). *GitHub—Openai/whisper: Robust Speech Recognition via Large-Scale Weak Supervision*. https://github.com/openai/whisper

Github. (2024b). *GitHub—Pallets/flask: The Python micro framework for building web applications*. https://github.com/pallets/flask/

Github. (2024c). *H/pytesseract*. https://github.com/h/pytesseract

Hevner, A. R. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems, 19*(2), 87–92.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly, 28*(1), 75–105. https://doi.org/10.2307/25148625

Hobert, S. (2019). Say hello to 'coding tutor'! Design and evaluation of a chatbot-based learning system supporting students to learn to program. *ICIS 2019 Proceedings*.

Hobert, S. (2023). Fostering skills with chatbot-based digital tutors – training programming skills in a field study. *i-com, 22*(2), 143–159. https://doi.org/10.1515/icom-2022-0044

Hobert, S., & Berens, F. (2024). Developing a digital tutor as an intermediary between students, teaching assistants, and lecturers. *Educational Technology Research and Development, 72*(2), 797–818. https://doi.org/10.1007/s11423-023-10293-2

Hobert, S., Følstad, A., & Law, E. L.-C. (2023). Chatbots for active learning: A case of phishing email identification. *International Journal of Human-Computer Studies, 179*, 103108. https://doi.org/10.1016/j.ijhcs.2023.103108

Hobert, S., & Meyer von Wolff, R. (2019). Say Hello to Your New Automated Tutor – A Structured Literature Review on Pedagogical Conversational Agents. *Wirtschaftsinformatik 2019 Proceedings*, 301–314.

Huang, W., Hew, K. F., & Fryer, L. K. (2022). Chatbots for language learning—Are they really useful? A systematic review of chatbot-supported language learning. *Journal of Computer Assisted Learning, 38*(1), 237–257. https://doi.org/10.1111/jcal.12610

Laugwitz, B., Held, T., & Schrepp, M. (2008). Construction and Evaluation of a User Experience Questionnaire. In A. Holzinger (Ed.), *HCI and Usability for Education and Work* (Vol. 5298, pp. 63–76). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-89350-9_6

The Tabler Authors. (2024). *Tabler/tabler*. https://github.com/tabler/tabler

The UEQ Team. (2024). *UEQ Data Analysis Tools*. https://www.ueq-online.org/Material/Data_Analysis_Tools.zip

trychroma.com. (2024). *chroma—The AI-native open-source embedding database*. https://www.trychroma.com

Weizenbaum, J. (1966). ELIZA—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45. https://doi.org/10.1145/365153.365168

Winkler, R., Hobert, S., Salovaara, A., Söllner, M., & Leimeister, J. M. (2020). Sara, the Lecturer: Improving Learning in Online Education with a Scaffolding-Based Conversational Agent. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–14. https://doi.org/10.1145/3313831.3376781

Winkler, R., & Söllner, M. (2018). Unleashing the Potential of Chatbots in Education: A State-Of-The-Art Analysis. *Academy of Management Annual Meeting (AOM)*.

Wollny, S., Schneider, J., Di Mitri, D., Weidlich, J., Rittberger, M., & Drachsler, H. (2021). Are We There Yet? - A Systematic Literature Review on Chatbots in Education. *Frontiers in Artificial Intelligence*, *4*, 654924. https://doi.org/10.3389/frai.2021.654924

MAL-E: Understanding Text-to-Image Generation

Silvia Joachim and Martin Hennecke

University of Würzburg, Computer Science Education

{silvia.joachim, martin.hennecke}@uni-wuerzburg.de

Abstract

Generative AI, particularly in image generation, has attracted a lot of attention in recent years. These technologies are here to stay. Understanding how they work demystifies them, showing that they are driven by algorithms, not magic. We present a learning and experimentation module for Unplugged Text-To-Image Generation, which we have called MAL-E. It explains several key steps in the process, starting with Tokenization, Embedding, and Positional Encoding. Students learn Masked Self-Attention, an important step of the Decoder-only Transformer. They also learn about Image Components Selection and Image Assembly to produce the final image. MAL-E provides an inclusive hands-on way to understand how pre-trained neural networks and transformers create images from text prompts.

Keywords: Artificial Intelligence · CS Unplugged · Inclusive Material · Generative AI · AI Education · Neural Networks · K-12 students.

Introduction

The ability to quickly create individual and professional images plays a crucial role in various aspects of life. With the right software, generating images from text is straightforward. For example, a prompt like “Draw a mushroom with stripes” directly produces an image of a mushroom with stripes. However, if you draw a mushroom yourself and then provide the software with a photo of that mushroom alongside the prompt “I want this exact mushroom, but with a striped pattern”, the resulting image will indeed show a striped mushroom. In many tools, the resulting image may differ from what one might have originally envisioned. To understand this, you need to understand at least a few basic ideas of the text-to-image generation. Incidentally, mushrooms are a recurring example in our materials for teaching various AI algorithms [3, 8, 9, 10], as they work very intuitively for almost all age groups.

In this paper, we introduce an unplugged approach designed to explain the concept of text-to-image generation using neural networks to an audience without a deeper background in AI, which includes high school students, university students without AI focus, and computer science teachers. The idea of teaching CS content without computers goes back to [4, 5] and has proved to be very successful in many areas. In the AI competency framework for students [14], unplugged learning materials are mentioned as one of the ways to support understanding AI. The training phase of the neural network is beyond the scope of this learning module, as our primary objective is to explain how the pre-trained model transforms text into images. Some AI-systems use only the decoder part of the transformer architecture. We do this also and simplify the steps involved in text-to-image generation to effectively teach the fundamental concepts. To ensure inclusivity for visually impaired students, we use materials with tactile features.

Unplugged Text-to-Image Generation

Understanding generative AI currently plays no role in the curricula of German schools (even if its application does). Initial work in CS education has focused on continuing character strings, sound sequences or hand drawings with Markov chains and making this functionality teachable [6, 15]. There are also examples of unplugged material here [11, 12]. [1] presents a training unit for teachers that also includes a technical explanation of text-to-image generation. Based on infographics, the unit primarily explains the diffusion model, while our learning module focuses on the Decoder-only Transformer architecture and emphasizes unplugged activities. It is challenging to count this as related work because the underlying technology and the generation processes differ significantly.

Of course, there are numerous academic articles on text-to-image generation (e.g. [17, 16]), well-known books [18] that at least outline this aspect, lecture manuscripts, etc. What these sources have in common is that they try to represent the generative process as well as possible and avoid simplification where possible. Since the aim of these sources is to present the state of the authors' work to other scientists or to explain to students with a particular interest in AI how they can develop such systems themselves, this approach is the right one. For our audience, such sources are usually too complex. Here a simplification is needed that preserves the essential functionalities of the real process but is understandable at the level of the audience.

Teaching Concept of MAL-E

MAL-E offers two levels of difficulty and includes individual and team activities. The name is derived from the German word for "paint", with its spelling paying homage to DALL-E [2], a practical tool in the field of text-to-image generation. One of our key challenges was designing a module that would exclude as few learners as possible. To achieve this, we opted for special unplugged materials. The game pieces feature tactile dice numbers, and the game board provides a stable base, allowing for the addition of Braille numbers to accommodate visually impaired students. We deliberately chose to work with only small numbers, and we kept everything within a two-dimensional framework. The transparent game boards and game pieces are available through the AI Experiment Kit [7]. If you want to do MAL-E without the desirable tactile properties, the material can also be realized in the form of printed worksheets. As the most important learning prerequisite, learners should already know how feedforward networks work, because they use simple pre-trained networks for MAL-E.

Activities in MAL-E

First, students divide into groups of four. Each group selects a slip of paper with a prompt. Different prompts, such as "Draw mushroom with stripes" or "Draw mushroom with flowers" can be chosen, or the same prompt can be selected multiple times. This prompt serves as the basis for the activities tokenization, embedding, positional encoding, masked self-attention, image components selection and image assembly.

Activity 1: Tokenization We consider the prompt "Draw mushroom with stripes". There are multiple ways to approach tokenization. We simply use the words of the prompt as tokens. In this activity there are game pieces with different words, from which the team selects the required tokens.

Activity 2: Embedding Neural networks require vectors as input. The goal of the Embedding activity is to assign a two-dimensional vector to each token. The students use a prepared coordinate system (figure 1 left) where various tokens are already placed. The closer the tokens are in meaning, the nearer they are positioned to each other. For example, the tokens "mushrooms" and "dots" are closer together than the tokens "mushrooms" and "stripes". For visually impaired students, only the tokens required for the specific prompt can be used, and instead of words, game pieces with the corresponding number of dots representing the position are utilized. Instead of coordinate axes, a rubber band can be stretched.

Activity 3: Positional Encoding The meaning of a prompt can change when the order of words is rearranged. For example, the prompts "Draw mushroom with stripes" and "Draw stripes with mushroom" have different meanings, even though they contain the same words, and would result in completely different images. Transformer models do not have a built-in sequence order for input, so positional encoding is added to represent the position of each token in the sequence. We simply add the vector (1,1) to the first token, (2,2) to the second token, and so on. This simplification ensures that we work only with small natural numbers while remaining in two dimensions, which allows us to represent the process enactively using game pieces for hands-on learning. The students read the coordinates for each of the used tokens from the left coordinate system in figure 1. The right coordinate system in figure 1 is initially empty. After performing the positional encoding, the students place their tokens at the corresponding positions in the right coordinate system.

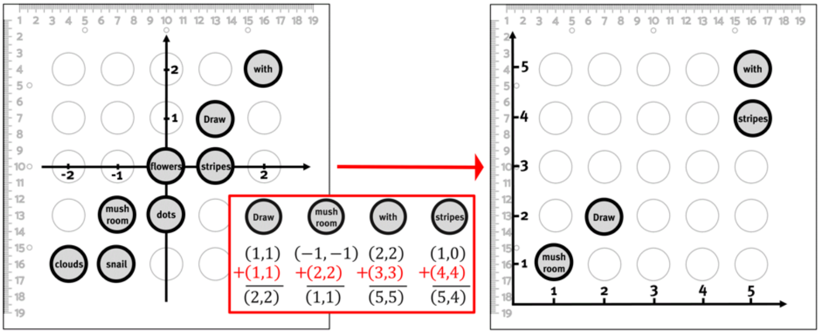


Fig. 1. Embedding and result after Positional Encoding for the selected prompt

Activity 4: Masked Self-Attention The unplugged materials are structured to visually demonstrate the computation of the Masked Self-Attention mechanism step-by-step. This is the central step in the text-to-image generation process, where the Decoder-only Transformer calculates the relationships between tokens in the sequence, ensuring that each token only considers the tokens that came before it. Masked Self-Attention means each token can only "see" the previous tokens, e.g. in the prompt "Draw mushroom with stripes" the token "mushroom" can see itself and "Draw", but not "with" and "stripes". Each student selects a token and takes one of four wooden tablets. Each tablet is equipped with the same game board and some game pieces with dice numbers. The tablets are large enough to allow the board to be shifted down during the activity.

The chosen token is placed in the top left corner of the board. Beneath the transparent board, there is a representation of three feed-forward networks with two input fields. The students places game pieces with the corresponding number of dots (representing the vector after Positional Encoding) in the fields at the top.

Query, Key, and Value are calculated using the neural networks. The activation function used in MAL-E is the identity function, which means the output of each neuron is the same as the combined input. These results are recalculated by each student for their token. The students place game pieces with the corresponding number of dots into the output fields, representing the results of the calculations. Each student performs the same calculations on their respective tablets, but each uses a different input vector. Figure 2 shows the results for the first two tokens. While each student works on their own, they can help one another, as the process is fundamentally the same for everyone.

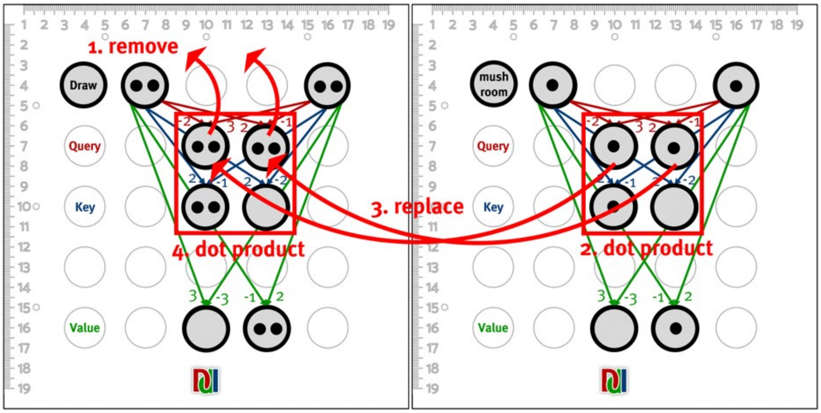


Fig. 2. Part of the process of Masked Self-Attention

The Query vector of a token represents the kind of information this token is looking for. The Key vector represents the kind of information that token holds. The alignment between the Query and Key vector can be mathematically calculated using the dot product. After each student has completed their game board, they place their tablets side by side in the order of the tokens in the prompt, forming a large square. The game boards are arranged diagonally, starting from the top left. The students now work together as a team. First, the first token “Draw” is considered. “Draw” can only see itself, so only the dot product between the Query and Key of token 1 is calculated. Next, the second token is considered. To do this, the game board of token 1 is placed next to the game board of token 2, as shown in figure 2. Token 2 considers both itself and Token 1. The game pieces representing the Query vector of Token 1 are no longer needed and are removed from the board. We now calculate how Token 2 perceives itself by computing the dot product between its own Query and Key vectors. Next, the game pieces from Token 2’s Query vector are placed in the fields for the Query vector of Token 1. This allows us to compute the dot product for Token 1’s board, giving us a result that represents how Token 2 perceives Token 1. To consider the third token, the two game boards are shifted down next to the board of token three. Now, only the Query game pieces from token three are needed, and the three dot products are calculated accordingly. The boards are then shifted down to consider token four, using only the Query game pieces from token four. The dot product can also be thought of as the angle between the original vectors pointing to the coordinates (x, y) of Query and (x, y) of Key. The students can use an Excel spreadsheet to apply the Softmax function to normalize the attention scores (dot products). Finally, the normalized attention scores are multiplied by the Value vectors to obtain the final vectors. For younger students and students with visual impairments, the Max function can be used instead of the Softmax function for certain prompts, allowing the final vectors to be calculated mentally.

Activity 5: Image Components Selection The previously explained steps assign a final vector to each token. For example, we obtain the vector (3,3) for the token “stripes”. The students therefore choose a box labeled (3,3). In this box they find different parts of the picture, which are transparent with engraved lines. Each of the picture parts has one of the markings 1, 2, 3 or 4. The students may now select a picture part from all the picture parts with the marking 1. They do the same with the picture parts marked 2 to 4. Once the students have chosen a picture part for all the tokens, they move on to the next activity.

Activity 6: Image Assembly The decoder generates the final image. The students overlay and arrange the transparent sheets side by side. They place all sheets labeled with the number 1 in the top right corner, number 2 in the bottom right, number 3 in the bottom left, and number 4 in the top left. Figure 3 shows how these transparent sheets can be combined to achieve the final composition of a new mushroom with stripes, giving the impression of a creative process.

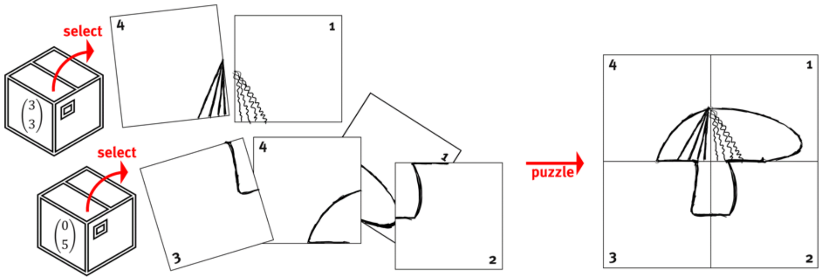


Fig. 3. Process of image component selection and assembly

Conclusion

The unplugged material presented here to explain text-to-image generation is part of a larger teaching sequence on AI. We found it necessary to create an enactive approach for this topic as well. Teachers should be aware of possible algorithmic biases [13]. By MAL-E, the picture parts in the boxes represent predefined components that the system selects based on its prior training data. If the boxes only contain certain parts or styles, the final output will be biased towards those, limiting the variety of images. Enactive materials have proven to be particularly effective in introducing new topics into the classroom.

MAL-E was tested with students and received very positive feedback. We conducted interviews based on self-assessment of their knowledge. All participants confirmed that MAL-E can effectively helps to understand how AI systems generate an image from a text prompt. MAL-E is an engaging and motivating learning module that introduces students to the workings of generative AI in image creation. It encourages students to understand how things work. Along the way, they learn about algorithmic thinking and the basics of neural networks. Further tests are planned as part of teacher training programs.

References

1. Ali, S., Ravi, P., Moore, K., Abelson, H., Breazeal, C.: A picture is worth a thousand words: Co-designing text-to-image generation learning materials for K-12 with educators. Proceedings of the AAAI Conference on Artificial Intelligence 38, 23260–23267 (03 2024), https://doi.org/10.1609/aaai.v38i21.30373

2. Amatriain, X.: Transformer models: an introduction and catalog (02 2023), https://doi.org/10.48550/arXiv.2302.07730

3. Andres, D., Joachim, S., Hennecke, M.: Den k-Means-Algorithmus verstehen: Mit Stift & Papier und BlueJ. Informatische Bildung in Schulen 2(1) (2024), https://doi.org/10.18420/ibis-02-01-06

4. Bell, T., Rosamond, F., Casey, N.: Computer Science Unplugged and Related Projects in Math and Computer Science Popularization, pp. 398–456. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), https://doi.org/10.1007/978-3-642-30891-8_18

5. Bell, T.C.: Teaching teachers to teach computer science - unplugged or plugged-in? Proceedings of the 2020 ACM Conference on International Computing Education Research (2020), https://api.semanticscholar.org/CorpusID:221035199

6. Hielscher, M.: SoekiaGPT - ein didaktisches Sprachmodell. Informatische Bildung in Schulen 1(1) (2023), https://doi.org/10.18420/ibis-01-01-04

7. Joachim, S.: Experimentiersatz Künstliche Intelligenz. MEKRUPHY GMBH (2023), https://mekruphy.com/de/ki

8. Joachim, S.: Experimentiersatz KÜNSTLICHE INTELLIGENZ - Experimentierheft. MEKRUPHY GMBH (2023).

9. Joachim, S.: Experimentiersatz KÜNSTLICHE INTELLIGENZ - Handreichung für die Lehrkraft. MEKRUPHY GMBH (2023).

10. Joachim, S., Hennecke, M.: Enaktive Bestimmung der Hyperparameter beim Entscheidungsbaum- und k-nächste-Nachbarn-Algorithmus. In: INFORMATIK 2023 - Designing Futures: Zukünfte gestalten, pp. 415–418. Gesellschaft für Informatik e.V., Bonn (2023), https://doi.org/10.18420/inf2023_47

11. Joachim, S., Hennecke, M.: Reinforcement-Learning enaktiv und inklusiv vermitteln. INFOS 2023 - Informatikunterricht zwischen Aktualität und Zeitlosigkeit (2023), https://doi.org/10.18420/infos2023-049

12. Lindner, A., Seegerer, S., Romeike, R.: Unplugged activities in the context of AI. In: Informatics in Schools. New Ideas in School Informatics: 12th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2019, Larnaca, Cyprus, November 18–20, 2019, Proceedings. p. 123–135. Springer-Verlag, Berlin, Heidelberg (2019), https://doi.org/10.1007/978-3-030-33759-9_10

13. Miao, F., Cukurova, M.: AI competency framework for teachers. UNESCO (2024), https://doi.org/https://doi.org/10.54675/ZJTE2084

14. Miao, F., Shiohira, K.: AI competency framework for students. UNESCO (2024), https://doi.org/https://doi.org/10.54675/JKJB9835

15. Mönig, J.: Snap!GPT – Bausteine für generative künstliche Intelligenz. Informatische Bildung in Schulen 2(1) (2024), https://doi.org/10.18420/ibis-02-01-04

16. Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., Chen, M.: Hierarchical text-conditional image generation with CLIP latents. ArXiv abs/2204.06125 (2022), https://api.semanticscholar.org/CorpusID:248097655

17. Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., Sutskever, I.: Zero-shot text-to-image generation. ArXiv abs/2102.12092 (2021), https://api.semanticscholar.org/CorpusID:232035663

18. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach (4th Edition). Pearson (2020), http://aima.cs.berkeley.edu/

Large Language Models as Domain-independent Dialogue Component for Intelligent Tutoring Systems – Teaching Concepts of SQL[1]

Adrian Voelker, Anna M. Thaler, Maximilian T. Summerer, and Ute Schmid

University of Bamberg, Germany

{adrian.voelker, anna.thaler, ute.schmid}@uni-bamberg.de
maximilian-timon.summerer@stud.uni-bamberg.de

Abstract

With the advance of generative pre-trained language models, new opportunities arise for domain-independent dialogue-based instruction in the context of intelligent tutoring systems (ITS). We propose an approach for combining large language models (LLMs) with domain-specific knowledge graphs to take advantage of the performance of LLMs in natural language processing tasks while still ensuring a faithful knowledge base. We present a prototypical implementation of an ITS for the structured query language SQL to explore the potential of combining LLM and domain specific knowledge graphs for dialogue based didactic intervention. While learning a programming language relies on both declarative knowledge about the language concepts and procedural knowledge for writing program code with respect to a specific task, in this paper we focus on ITS support for acquisition of declarative knowledge. In our implementation LLMs are employed for semantic parsing relating student's questions to the addressed knowledge elements of the domain model as

[1] The work reported in this paper, has been supported by Bund-Länder Förderinitiative "Künstliche Intelligenz in der Hochschulbildung", project "VoLL-KI: Data-based analyses, explanations and aids", project number 16DHBKI091.

well as for feedback formulation based on retrieved information from the knowledge graph. Likewise, knowledge diagnosis is realized by semantic parsing of student answers and mapping them against the LLM output. We conducted a first exploratory evaluation using the LLM Mixtral 8x7B Instruct in combination with our SQL knowledge graph. Exploring different prompting strategies, the best strategy resulted in 90% correct diagnosis of student answers.

Keywords: Intelligent Tutoring Systems, Large Language Models, Knowledge Graphs, SQL, Declarative Knowledge

Introduction

Programming education has been one of the most implemented domains in intelligent tutoring systems (ITS) [11]. Over time many different systems for multiple programming languages [5], such as LISP [2], Python [18], and the Structured Query Language (SQL) [10] were developed. SQL in particular is a widely used language for efficient data retrieval and manipulation in many business processes that is relevant far beyond the field of computer science and therefore of interest to a broad audience from computer science undergraduate students to casual data-oriented users.

Learning a programming language relies on both declarative knowledge about the language concepts and procedural knowledge for writing program code with respect to a specific task. Typically, ITS for programming have a strong focus on procedural knowledge [13]. However, declarative knowledge about programming concepts and their relationships is not only relevant as a knowledge base for the acquisition of programming skills in the given domain, it also substantially enhances the transfer between tasks of different sub-skills (such as programming versus understanding a program) that share the same declarative knowledge base [17]. Therefore, declarative knowledge about concepts and the relationships between them can play a crucial part in supposedly purely procedural tasks. Errors in programming tasks could occur due to existing misconceptions while performing the (procedural) task or due to missing declarative knowledge about the underlying concepts necessary to correctly solve it.

Thus, we encourage to consider conceptual knowledge within ITSs for programming. In this paper, we present our current work on the declarative part of an ITS for learning the programming language SQL, which we afterwards combine with procedural knowledge tasks for a flexible switching between the two knowledge types and thus a more accurate student knowledge diagnosis.

One major restriction in the development of ITSs is that the architecture is focused on specific teaching domain and the realized teaching strategies. Typically, ITS require a large number of resources for their development [13]. An often-laborious part of the implementation concerns the domain-specific methods of natural language processing (NLP) for a natural dialogue as form of interaction with the students [6]. The latest developments in large language model (LLM) research promise great potential to move towards a domain-independent method to solve NLP tasks. The development effort for parts of the tutoring and communication module could, therefore, be significantly lowered as a result.

In this paper, we present a prototypical implementation of an ITS for SQL to explore the potential of combining LLM and domain specific knowledge graphs for dialogue based didactic intervention to enable students asking questions while also enabling the ITS with the capability to check and correct given answers. Hereby, we want the LLM to assist in the identification of distinct question-types task and evaluate a given answer. The used LLM is fine-tuned for the first task and prompting techniques are used for the latter. In the following, we first discuss related work, and then describe relevant system components. We show results of a preliminary evaluation exploring different prompting strategies. We conclude with an outlook towards the next steps of extending the implementation and a more comprehensive evaluation.

Related Work

The use of LLMs has been investigated in the context of annotating student actions to corresponding tutoring actions as a way to reduce the resources required for annotations by pedagogic experts [20]. However, the

LLM's helpfulness of feedback in particular has been found to be quantifiably worse compared to human teachers [19]. First attempts to evaluate the effectiveness of LLMs as a sole pedagogical instructor compared to a human teacher have been made [19], but standardized and comprehensive criteria for the evaluation of conversational agents within intelligent teaching are still missing.

Knowledge-based ITS for declarative tasks often use a knowledge graph (KG) as their knowledge base. While a KGs' explicit knowledge representation ensures high precision inference and reasoning [16], LLMs may produce misleading, untrustworthy or incorrect responses, such as hallucinations [7], due to their probabilistic nature [14]. Their output could amplify biases from their training data and they lack in the ability to handle numerical values [14]. Especially when interpretability and explainability is required - which is the case for most ITS - KGs can explicitly represent the structure and relationships between entities [14] and therefore form the basis for the tutoring module's feedback. A sensitive topic regarding the education of students is misinformation. As interacting students are still learning a subject, misinformation might be harder to detect and thus can have a higher negative impact on the learning process compared to experts using LLMs in the domain of expertise. In order to reduce the risk for non-trustworthy responses, we argue that relying on an LLM's parametric knowledge alone is currently not suitable as a knowledge base for ITS.

This might lead to the question of why we want to incorporate LLMs in ITS in the first place. The biggest advantage of pre-trained language models lies in their great performance in the field of language processing, general knowledge and generalizability [16]. Neuro-symbolic methods, therefore, seem to be a promising approach to simplify the development process for ITS. In recent literature were three frameworks for unifying KGs and LLMs defined. KGs can enhance LLMs and could e.g. be used to detect LLMs' hallucinations [14,16] and to improve the reasoning capabilities of a LLM for complex logical problems [15]. LLMs could augment KGs with their natural language processing ability, which we currently view to be most promising for the domain of education. The third framework in their roadmap lies in a full synergy of both methods.

Similarly, Bianchini et al. (2024) argue that to adequately (re)solve complex linguistic tasks, what we herein consider a dialog during a didactic intervention of the ITS or the parsing and contextualizing of questions towards the Expert System, a knowledge graph is essential [3]. We further on share the idea, that solely combination of LLMs with Retrieval Augmented Generation (RAG) on diverse knowledge sources such as a documentation for SQL, is not sufficient for an expert system, since eventually the enriched LLM component leads to hallucinatory output, even though a knowledge base was supplied [3].

A hybrid representation of both parametric knowledge (LLMs) and explicit knowledge in the form of KGs, is already being discussed in literature [14,3]. This combination, however, still requires further research and evaluation and more contextualization towards ITS for SQL. Therefore, we first want to explore how LLMs can enrich a knowledge-based tutoring system by simplifying a natural language dialogue and answer validation without relying solely on LLMs and RAG as knowledge base.

System Components

In the following section we highlight the fundamental system components of our ITS. In this paper we want to especially focus on the aforementioned declarative knowledge. Since we consider also procedural knowledge of SQL as a indispensable requirement, we plan our system to be integrable with further components, that are not highlighted in more detail in this paper.

3.1 The Declarative Domain Knowledge Base for SQL

As explicit and declarative knowledge base we employ a semantic network (a KG) on basic SQL concepts based on previous work by Zhou [22]. Zhou identified two types of declarative knowledge within SQL: the natural and the abstract form representation. The natural form includes at least one property about the concept in natural language, e.g. a description about the function of the concept, whereas the abstract form describes the relationship between concepts [22].

Labelled property graphs are highly suitable for representing both forms in one consistent knowledge representation. Each node represents a concept, e.g. the select statement, and the edges between the nodes represent the relationship between concepts, e.g. "has a", "type of", "kind of", "part of" or "set of". Using domain-independent expressions for the edge labels builds the foundation for a domain-independent interaction with the LLM as intermediate component (see Figure 1). We expect that theses generic labels allow using the same inference mechanisms (queries to the graph database within the domain module) and the same prompts for the LLM, even if the entire domain (the graph database) changes. Each node (concept) holds the properties relevant for the comprehension of the concept, e.g. a description of the function, the correct notation, or allowed input (if applicable). These properties can be used for the question generation for a single concept as well as for the comparison between concepts.

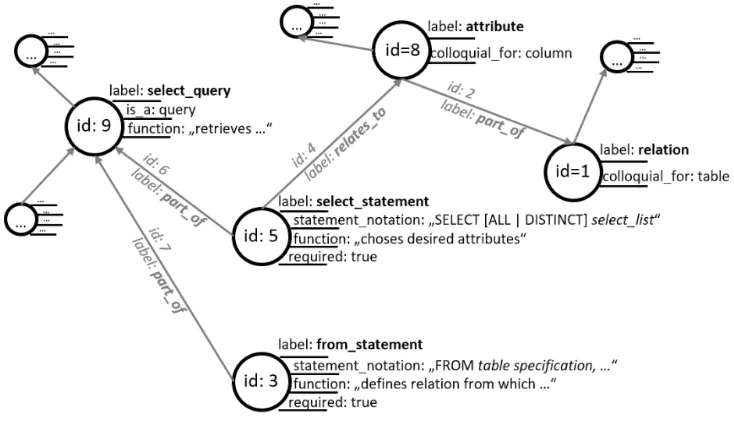


Fig.1. Excerpt and simplified representation of a possible semantic network for SQL knowledge as basis for domain-unspecific inference mechanisms.

3.2 Possible Application Points for LLM

We aim at combining LLMs and KGs as part of an ITS within their scope of expertise. LLMs offer great potential for a nearly effortless built, domain independent and dynamic natural language dialogue, but we do not rely on its parametric, possibly incorrect knowledge. KGs, in contrast, ensure correct inference mechanisms as well as domain-specific and explicit knowledge that provides the base for explainability and feedback formulation. Possible starting points that we see for the integration of LLMs within a simplified version of an ITS architecture are shown in Figure 2. The integration of LLMs into ITS can lead to a further separation of domain-specific modules such as the knowledge representations in the domain or student module from domain-independent natural language processing mechanisms. These could include but are certainly not limited to the extraction of intentions by the student, the answer validation (diagnostic comparison operation between expected and given solution), and the student interaction such as the formulation of feedback or follow-up questions. In the following, we want to outline these mechanisms as approaches for these integration points in the way we are currently implementing them into our system for intelligent SQL tutoring.

1. ***Student Input Translation:*** All queries to the semantic network (labelled property graph) are mediated by controlling components of the ITS. There are approaches that aim to translate natural language directly to a query [14], however, we want to ensure that the inference mechanisms on the KG are considered correct and we – the developer – know the attributes of the semantic network best. Therefore, when a student asks a question in natural language to the ITS, that question is not directly translated to a query (in our case CYPHER within Neo4J [12]), but the LLM is used to translate the student's question to categories of questions (e.g. concerning properties of concepts, or relationships between concepts). By retrieving the essence of a student's question with the help of a LLM, we can formulate our query to the graph database and, therefore, provide higher independent components by separating the dialogue from accessing the KG itself. This is a key point for the transfer of the same methods to other domains as knowledge base.

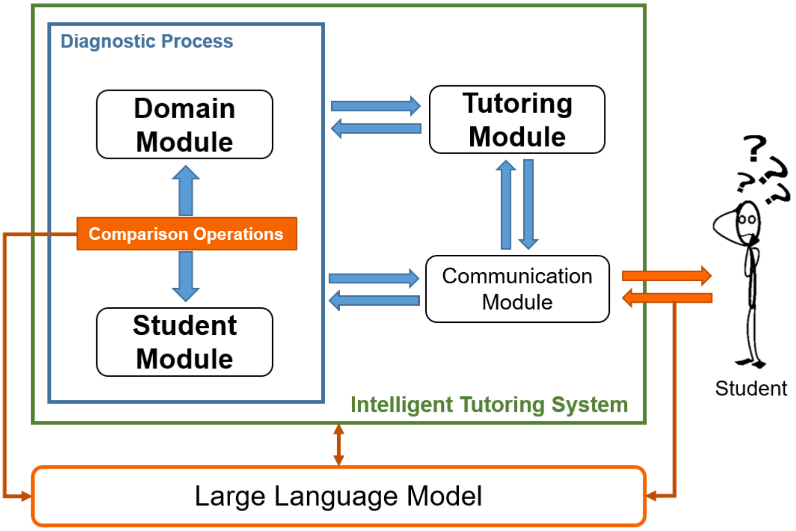


Fig.2. Classic Four-Component Architecture of ITS with possible integration points for LLM marked in orange.

2. ***Answer Validation as Comparison Operation:*** The LLM also offers a flexible way to overcome classical steps of other NLP methods such as the equivalence of specific terms in the sense of grammar or spelling mistakes, or synonyms, but also domain-specifically equivalent terms. In the domain of SQL, for example, students might refer to a relation as table or to a tuple as row in a table. The task of the LLM in this case is to compare the expected answer from the labeled property graph (KG) with the answer given by the student. This way, the lack of domain-specific knowledge of the LLM does not come into play, as the explicit and faithful knowledge from the KG is used for reasoning and serves as expected expert response to compare to the student's reply. The LLM's generated comparison can be used to update the student model accordingly and serves as starting point for analysis when the answers are not found to be completely the same.

3. ***Feedback Formulation:*** Another integral point lies in the formulation of ITS information to natural language for easier understanding by the student. This can be applied for domain-specific explanations based on the KG's data within the domain module, but also didactic strategies within the tutoring module, the LLM is prompted to simply generate natural language feedback from a more restricted and formalized form from within the ITS to present it to the student.

The information from the student model about the current state of comprehension can be used within the prompt to adapt the complexity of the natural language feedback to the student's current diagnosed knowledge.

3.3 Didactic Intervention

The KG explicitly stores data about the concepts and the concepts' relationship and therefore also holds information about their similarities. This constellation can be exploited to create appropriate didactic interventions, e.g. a dialogue following the Socratic instructional strategy as a form of scaffolding [1]. Given the student has already studied (parts of) the subject, when an incorrect answer is given to one concept, the student will be asked similar questions about a similar concept. When knowledge for this similar concept is available, it can serve as a starting point for the student to decode and recall information about the originally disremembered concept. For example, if a student does not recall the description of the aggregate function 'SUM', one possible similar concept for the Socratic questioning might be to ask what 'COUNT' does. This not only enhances memory recall to evoke already acquired but currently not accessible knowledge to the student but also serves as assessment for the student knowledge model. The similarity scores between concepts can be built upon several factors including hierarchical structures such as super- or subconcepts (based on closeness of concepts within the KG structure), but also the concept's properties.

Preliminary Evaluation

We tested and evaluated language model integration in the first two areas described in Sect. 3.2. When users ask questions to retrieve information, a language model is used to match their question to pre-defined question categories:

- **Property question.** (Example: What is %s's %s?)
- **Relation question.** (Example: What is the relation between %s and %s?)
- **Relatives question.** Example: To what node(s) is %s connected via %s relation?)
- **Similarity question.** (Example: What similarities do %s and %s have?)
- **Difference question.** (Example: What are the differences between %s and %s regarding %s?)

Furthermore, when interrogating a user, a language model is used to compare the student answer with the correct solution provided by the KG.

Language Model for Question Category Classification

An attractive application of a language model is as a mediator between user question and information retrieval from the KG. In our ITS structure knowledge is retrieved via predefined question categories, ensuring that no ill-formulated queries are forwarded to the KG. However, determining the most suitable question category for a user question is not trivial. Users can use a variety of formulations for the exact same question. Here, a language model can be used to 'translate' the user's question into one of the pre-defined question categories. Since general language models did not deliver good results in this context, an instance of TinyLlama [21] was fine-tuned on a dataset containing 250 entries with possible user questions and their corresponding question categories. The resulting model proved useful for cases where the user question was either similar to the formulation of the question category itself or similar to the entries in the dataset. However, the more a user question diverged from the expected result, the less frequently the language model delivered the correct match. To address this problem, a different, better performing language model could be used as a basis, a more extensive and more diverse dataset could be used to fine-tune this model. Furthermore, constraining the model to a grammar by utilizing the Backus–Naur form (BNF) has not shown useful results so far. However, this could be further explored.

Language Model for Student Answer Validation.
For the ITS to function effectively, it is crucial to evaluate the correctness of students' answers. While the KG provides only one correct answer, students may express the correct solution using formulations that are completely different from the stored information. This variability makes it difficult for the ITS to determine whether the student answered correctly. To address this issue, using a language model appears promising, as it could grasp the semantic meaning of a user's answer, compare it to the solution provided by the KG, and decide whether or to what degree the student answered correctly. For an initial implementation of this feature, an instance of the Mixtral 8x7B Instruct Model [9,8] was used. A sample dataset was created, comprising ten distinct questions. For each question, the dataset contains three possible user answers: one clearly correct, one clearly incorrect, and one either correct or incorrect but not immediately recognizable. This setup results in a total of 30 entries, with 15 to be classified as correct and 15 to be classified as incorrect (see Table 1 for exemplary questions). Four different prompts were compared: Two longer prompts with examples of the expected output and two prompts containing only a brief instruction of the task.

Remarkably the prompt with a brief instruction using a courtesy form (see Figure 4) was leading to better results than longer and more specific prompts (see Figure 4)). In the first evaluation, the prompt in Figure (4) without examples resulted in an accuracy of 90% on the described dataset. In the future, different prompts, system prompts, and fine-tuned language model could be tested on more extensive datasets, preferably obtained from actual user answers.

Limitations

These implementations still come with several limitations. So far, only a handful of language models have been evaluated, and the data on which they were fine-tuned and evaluated is still limited. Additionally, no data has been collected from actual user-system interactions. Furthermore, it is still unclear whether the use of question categories is the best way to mediate between user questions and the KG.

Table 1. Exemplary test data for evaluating the match between LLM and KG category.

Tutor question	**LLM match to KG**	**Student answer**	**classification**
What is the relation between equal and comparison predicate?	TYPE_OF	type of	correct
To what node(s) is equal connected via TYPE_OF relation?	comparison predicate	to comparison predicate	correct
To what node(s) is relation connected via HAS_A relation?	relation name, cardinality, degree, domain, PK, FK	to relation name, degree, cardinality, PK, FK, and domain	correct
To what node(s) is equal connected via TYPE_OF relation?	comparison predicate	it is a type of aggregate function	incorrect
To what node(s) is equal connected via TYPE_OF relation?	comparison predicate	select	incorrect
…	…	…	…

“Please decide whether the student answer is correct, based on the correct solution.”

Fig.3. Best performing prompt with the Mixtral 8x7B Instruct Model

```
Your task is to determine whether a student answer is correct in relation to the solu-
tion and the question.

####
Here are some examples:

Question : To what node( s ) is avg connected via TYPE_OF relation ?;
Correct Solution: aggregate function;
Student answer: average is a type of aggregate function;
Classification: correct

Question: What is the relation between where and group by?;
Correct Solution: where–>COMES_BEFORE–>group by ;
Student answer: group by comes before where;
Classification: incorrect

Question: What is the relation between where and group by?;
Correct Solution: where–COMES_BEFORE–>group by;
Student answer: where comes before group_by ;
Classification: correct

Question: What is the description of avg ?;
Correct Solution: Calculates the average value of a numeric col-
umn;
Student answer: avg calculates the average of the columns ;
Classification: correct

Question: What is the description of avg ?;
Correct Solution: Calculates the average value of a numeric
column;
Student answer: avg tells me more about the average time a query takes to execute;
Classification: incorrect

Question: What is the relation between sum and aggregate function?;
Correct Solution: sum–TYPE_OF–>aggregate function ;
Student Answer: sum is a type of aggregate function;
Classification: correct

Question: What is the relation between sum and aggre-
gate function?;
Correct Solution: sum–TYPE_OF–>aggregate function ;
Student Answer: sum is equivalent to aggregate function;
Classification: incorrect
####
```

Fig.4. Less performing prompt with the Mixtral 8x7B Instruct Model

Discussion and Further Work

In this paper we show that LLMs can be combined with labelled property graphs to enable teaching of declarative knowledge for programming languages such as SQL. However, our current work builds the basis for domain-independent methods that can be transferred to other labelled property graphs as knowledge base given the same edge labels are used. Exploiting LLMs for usually very laborious method development for NLP tasks enhances the separation of domain specific knowledge and the student interaction in natural language.

Since the shown preliminary evaluations show promising results, we are currently assessing the performance for the answer validation by comparing different open-source models, and experiment with prompt engineering methods for more accurate and stable results. Furthermore, use-case specific evaluations should be conducted on how hallucinations of the LLM are more suppressed and hindered in this approach than without using the KG in the inference process. Adapting LLMs to the method of Automatic Short Answer Grading and the respective development pipeline [4] to adequately evaluate student responses in our ITS sounds promising for future research on answer validation in our context.

In the future we aim to combine this declarative knowledge base with procedural knowledge by mapping the concepts of our semantic network to specific tasks that require those underlying concepts. This should allow a flexible switch between testing for applied skills in SQL on specific databases and the underlying declarative concepts as prerequisite or supplement to the required skills for the task. Exploring the possibility to incorporate LLM into the tutoring module e.g. to select didactic strategies, seems to be a promising new integration point. However, to our knowledge, LLMs have not yet shown results comparable to suitable strategies proposed by human teachers, yet.

References

[1] Alshaikh, Z. et al.: Experiments with a socratic intelligent tutoring system for source code understanding. In: The Thirty-Third International Florida Artificial Intelligence Research Society Conference (FLAIRS-32). (2020).

[2] Anderson, J.R., Skwarecki, E.: The automated tutoring of introductory computer programming. Commun. ACM. 29, 9, 842–849 (1986). https://doi.org/10.1145/6592.6593.

[3] Bianchini, F. et al.: Enhancing Complex Linguistic Tasks Resolution Through Fine-Tuning LLMs, RAG and Knowledge Graphs (Short Paper). In: Almeida, J.P.A. et al. (eds.) Advanced Information Systems Engineering Workshops. pp. 147–155 Springer Nature Switzerland, Cham (2024).

[4] Burrows, S. et al.: The Eras and Trends of Automatic Short Answer Grading. International Journal of Artificial Intelligence in Education. 25, 1, 60–117 (2015). https://doi.org/10.1007/s40593-014-0026-8.

[5] Crow, T. et al.: Intelligent tutoring systems for programming education: a systematic review. In: Proceedings of the 20th Australasian Computing Education Conference. pp. 53–62 Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3160489.3160492.

[6]. Graesser, A.C. et al.: AutoTutor. In: Applied natural language processing: Identification, investigation and resolution. pp. 169–187 IGI Global (2012).

[7] Ji, Z. et al.: Survey of Hallucination in Natural Language Generation. ACM Computing Surveys. 55, 12, 1–38 (2023). https://doi.org/10.1145/3571730.

[8] Jiang, A.Q. et al.: Mistral 7B, https://arxiv.org/abs/2310.06825, (2023).

[9] MistralAI: Mixtral 8x7B Instruct, https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1, (2023).

[10. Mitrovic, A.: An Intelligent SQL Tutor on the Web. International Journal of Artificial Intelligence in Education. 13, 2–4, 173–197 (2003).

[11] Mousavinasab, E. et al.: Intelligent tutoring systems: a systematic review of characteristics, applications, and evaluation methods. Interactive Learning Environments. 29, 1, 142–163 (2021). https://doi.org/10.1080/10494820.2018.1558257.

[12] Neo4J Inc.: Neo4J, neo4j.com, (2024).

[13] Nkambou, R. et al. eds: Advances in Intelligent Tutoring Systems. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13451-7.

[14] Pan, J.Z. et al.: Large Language Models and Knowledge Graphs: Opportunities and Challenges, http://arxiv.org/abs/2308.06374, (2023).

[15] Pan, L. et al.: Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning, http://arxiv.org/abs/2305.12295, (2023).

[16] Pan, S. et al.: Unifying Large Language Models and Knowledge Graphs: A Roadmap, http://arxiv.org/abs/2306.08302, (2023). https://doi.org/10.48550/arXiv.2306.08302.

[17] Pennington, N. et al.: Transfer of Training Between Cognitive Subskills: Is Knowledge Use Specific? Cognitive Psychology. 28, 2, 175–224 (1995). https://doi.org/10.1006/cogp.1995.1005.

[18] Rivers, K., Koedinger, K.R.: Data-Driven Hint Generation in Vast Solution Spaces: a Self-Improving Python Programming Tutor. International Journal of Artificial Intelligence in Education. 27, 1, 37–64 (2017). https://doi.org/10.1007/s40593-015-0070-z.

[19] Tack, A., Piech, C.: The AI teacher test: Measuring the pedagogical ability of blender and GPT-3 in educational dialogues. arXiv preprint arXiv:2205.07540. (2022).

[20] Vujinović, A. et al.: Using ChatGPT to Annotate a Dataset: A Case Study in Intelligent Tutoring Systems, https://www.techrxiv.org/articles/preprint/Using_ChatGPT_to_Annotate_a_Dataset_A_Case_Study_in_Intelligent_Tutoring_Systems/23617551/1, (2023). https://doi.org/10.36227/techrxiv.23617551.v1.

[21] Zhang, P.: TinyLlama, https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0, (2023).

[22] Zhou, G.: Towards designing a knowledge-based tutoring system: SQL-tutor as an example. New Jersey Institute of Technology (1996).

Diagnosing Algorithms by Abduction from Manual Simulation

Moritz Bayerkuhnlein and Diedrich Wolter

Institute for Software Engineering and
Programming Languages
University of Lübeck, Germany
{firstname.lastname}@isp.uni-luebeck.de

Abstract

Taking a conceptual idea and turning it into a precise algorithm is at the heart of computational thinking. However, novice programmers often struggle when their code does not behave as *they* intended. This paper identifies problems in pseudocode algorithms based on deviations from observations that could be gathered during manual simulations by the programmer. By applying model-based diagnosis to the faulty pseudocode, informed by manual simulation, we locate errors and suggest fixes. The diagnosis results in the identification of a specific location in the algorithm and provides an output description for the faulty part that matches the programmer's intent during the manual simulation, thus aiding in debugging.

Introduction

Creating an algorithm starts with a rough idea that leads to a detailed executable implementation. Inexperience, misconceptions, or even a slight lapse in concentration can cause a programmer to get the details wrong, leading to a faulty implementation. Especially for novices, this can be a barrier to entry into programming in general.

A human programming tutor is adept at understanding both the written program and the student's intentions. Program comprehension involves developing a mental representation of the program, to identify any errors and determine how to fix them. When pointing out a particular error in the source code, the tutor will usually provide an *explanation* of how a particular line led to an unintended action, and what the correct action should have been to achieve the expected result.

Techniques from Artificial Intelligence (AI) (Shapiro, 1982; Wotawa et al., 2002) and Automated Debugging (Dovier et al., 2005; Weiser, 1984; Zeller and Hildebrandt, 2002) can provide assistance and help to *diagnose* suspicious lines in the source code. However, these techniques are aimed at experienced programmers who can take measures to fix the code once the fault is located.

Novices, particularly in a practice setting, may need more help to determine why a certain statement caused problems, especially when they have an inadequate *mental model* (Johnson-Laird, 1989) of the language or the data structure used and regard the algorithm produced as correct (Chmiel and Loui, 2004).

Here, we use model-based diagnosis techniques to diagnose faulty implementations based on the programmer's intent to master a given programming exercise. To do this, we trace a manual simulation of dominant data structures to capture the intermediate steps the programmer intended their program to go through. With this approach, we aim for a mental model aware fault diagnosis, where a source-code location that is identified as potentially faulty gets justified and aligned with the expected program behavior.

Preliminaries

When debugging software, we find discrepancies by testing the potentially faulty program Π with the correct input and output specifications (I, O). Fault localization techniques that rely on source code execution, such as Spectrum-Based Fault Localization (SFL) (Abreu et al., 2009), highlight code locations based on their co-occurrence with unexpected outputs. While these methods are efficient, they cannot isolate the underlying reasons why a statement may have produced an unexpected result.

The *inference to the best explanation* of an unexpected outcome, so-called abduction can reveal possible reasons using AI techniques (Magnani, 2011). In *Model-Based Diagnosis* (MBD) (Reiter, 1987), a system model, made form a set of connected components, predicts behavior based on inputs and is restricted by observed outputs. When a failure occurs, the model identifies discrepancies between the expected and actual output. A diagnosis is the set of components that, if assumed faulty, would *explain* the failure.

Formally, our goal is to identify the component of the source code $COMP$ in the form of a statement or a group of statements that could have caused the mismatch by labeling it *abnormal* (ab) (Console et al., 1993; Wotawa et al., 2002).

More formally this is specified by Reiter's Consistency-Based Diagnosis (Reiter, 1987), here using notation from Model-Based Software Debugging (Wotawa and Dumitru, 2022) using constraint solvers, where $M(\Pi)$ and $\mathrm{M(I,O)}$ is the translation of a program Π and test case $(\mathrm{I,O})$ into a logical model and reasoned about using a constraint solver.

Definition (Consistency-Based Diagnosis Problem): Given a program model $\langle M(\Pi), M(I,O), COMP\rangle$, compute all minimal sets $\Delta \subseteq COMP$ of abnormal components such the following expression is satisfied:

$$M(\Pi) \cup M(I,O) \cup \{ab(c) \mid c \in \Delta\} \cup \{\neg ab(c) \mid c \in COMP \backslash \Delta\}$$

Imperative programming languages and algorithms use variables that collectively represent the *state* of a program and establish dependencies (Weiser, 1984). Datatypes restrict the possible values of these states to a (finite) domain. Therefore, any diagnosis Δ must also find a satisfactory assignment for the affected variables; otherwise there would be no fix for a single statement (Bayerkuhnlein and Wolter, 2024).

Representing Algorithms for Diagnosis

In previous work, we modeled system descriptions using test cases that target individual functions as program components (Bayerkuhnlein and Wolter, 2023). For algorithmic diagnosis, the model structure in $M(\Pi)$ is based on the program's control flow.

Given an imperative program Π we parse its operations and control structures to create a complete flow graph $G(\Pi) = \langle COMP, E \rangle$, where nodes $COMP$ correspond to diagnosable statements. As control structures we consider *if* statements for selection and *while* loops for iteration; function and recursive calls are left for future work. Directed edges E represent the control flow of Π. The figure below illustrates this representation.

Each flow line in E marks a *state transition* during the execution of Π. In terms of a system description, this is how statements (components) are connected. Representing these transitions as relations allows inversion and reasoning about alternatives (Ross, 1997).

```
Algorithm 1: SearchBST
   Input: root: TreeNode, key: int
 1 node ← root;
 2 while node ≠ ∅ do
 3     if node.val = key then
 4         return node;
 5     else if key < node.val then
 6         node ← node.left;
 7     else
 8         node ← node.left;
 9 return ∅;
```

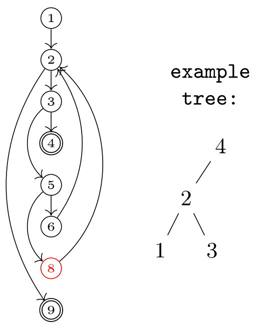


Figure 1: Algorithm with error and corresponding Control Flow-Graph

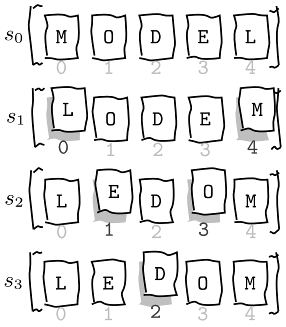
All variables used within Π are represented by their value $Var(\Pi)$ in a structure that represents the current state of the program, e.g. the shown algorithm *SearchBST* in Figure above has states of form $s = \langle root, key, node \rangle$.

Each node in G represents a statement in Π and is a component of $COMP$, making it a potential part of a diagnosis. Every statement defines behavior for input and output states, affecting the program pointer pc and potentially altering variable values $Var(\Pi)$. Transitions reflect control-flow manipulations, such as *if* statements. If a manipulation constrains state transitions to the point of impossibility, a conflict arises. This conflict is resolved by marking the statement as *abnormal*, thus identifying a diagnosis. In the above example, any right subtree cannot be found since line 8 updates the *node* variable incorrectly. To account for an observation (I, O) modeling *SearchBST(example_tree, 3) = node 3* instead of $\emptyset$, a fix of line 8 must transition node to node 3.

This approach presents challenges, as *any* state transition might be attributed to abnormal components, making it difficult to explain their behavior logically. The issue is worsened when loops are involved, as affected statements run multiple times. Inducing potential values along a sequence, such as program execution, faces under-constrained issues; abnormality at the very end of a program could explain any test result, regardless of prior actions. In practice, model-based software debugging is typically used to identify minor issues resulting from an experienced programmer's oversight (Wotawa and Dumitru, 2022).

Diagnosing from Manual Simulation

There are several conceptual solutions to a programming problem, each with various implementations. A test case cannot fully capture the programmer's intent, especially for algorithms with a single output per input, making internal workings opaque. Correct implementations appear identical in test cases, though they may differ in implementation. Faulty implementations can hide problems until later in the execution.



s_0 M O D E L 0 1 2 3 4
s_1 L O D E M 0 1 2 3 4
s_2 L E D O M 0 1 2 3 4
s_3 L E D O M 0 1 2 3 4

Algorithm 2: ReverseString

```
  Input: word: String
1 left ← 0;
2 right ← len(word) - 1;
3 while left < right do
4     word[left] ← word[right];
5     word[right] ← word[left];
6     left ← left +1;
7     right ← right −1;
8 end
```

Figure 2: String Reversal Manual Simulation and faulty Algorithm

Running Example (Reversing Strings I). *The student is tasked with implementing a procedure to reverse strings. Possible strategies include: (i) Copy the string to another array starting from the end, (ii) Swap elements in pairs, working towards the middle, or (iii) Use a stack to push and pop elements back into the array. Choosing strategy (ii), a student might find that manual simulation with the input string "MODEL" correctly results in "LEDOM". However, the implementation unexpectedly produces "LEDEL", indicating a discrepancy from the manual simulation. To suggest a correction, a tutor could pinpoint lines 4 and 5, noting that the implementation failed to execute the expected swapping behavior due to prematurely overwriting the left side.*

Solving tasks such as the one described is inherently a creative process (Knobelsdorf and Romeike, 2008). It involves first devising a general solution to the computation to be performed and then expressing that solution in a programming or pseudo-code language. People also enjoy developing *their own* solutions (Sharmin, 2021).

For absolute novices, an Intelligent Tutoring System (ITS) with model tracing, which uses a cognitive model to follow a student's problem-solving steps, provides helpful feedback by tracking and offering corrections (Anderson, 1993). However, modeling the vast domain of programming is challenging and can limit intermediate students by enforcing overly prescriptive solutions (Johnson and Soloway, 1985).

The aim of this approach is to respect the creative integrity of the student and to allow for multiple or varied strategies while still giving them some guidance to achieve their goal. Assisting novice programmers goes beyond finding and fixing bugs, but to identify and resolve the exact problem that caused the discrepancy between the conceptually intended program, their mental model, and the realized program.

Formally, we define the intention of a programmer as $\langle\hat{\Pi},\hat{\Sigma}\rangle$, where $\hat{\Pi}$ represents the mental model of the program and $\hat{\Sigma}$ represents the conceptual semantics, i.e., how the programmer thinks the components $c \in COMP$ operate. In contrast, the implemented program is denoted as $\langle\Pi,\Sigma\rangle$, where Π is the executable program and Σ represents the semantics of the programming language. Behavior of an individual component is denoted $\Sigma(c) \in \Sigma$.

Ideally, both $\hat{\Pi}$ and Π should exhibit identical behavior, as $\hat{\Pi}$ is the conceptualization of Π. However, discrepancies in behavior can arise due to implementation errors or misunderstandings of the program semantics (Chandra et al., 2024). Our objective, upon identifying such discrepancies, is to isolate components Δ within Π that deviate from the intended behavior $\langle\hat{\Pi},\hat{\Sigma}\rangle$. We aim to propose suitable alternatives $\hat{\Sigma}_\Delta = \{\, \hat{\Sigma}_\Delta(\mathrm{c}) \mid \mathrm{c} \in \mathrm{COMP}\,\}$ to align the program with its intended behavior, facilitating program repair or further analysis. The feasibility of synthesizing a program repair Π' that implements the proposed behavior $\hat{\Sigma}_\Delta(\mathrm{c})$ using Σ is not addressed here.

A conceptual program $\hat{\Pi}$ is low fidelity, representing only the conceptually relevant dominant data structures D during manual simulation. Observations from the manual simulation are represented as partial state descriptions, focusing solely on the most relevant data manipulated during algorithm execution. The remaining variables relevant to $M(\Pi)$ act as scaffolding, bridging the gap between conception and implementation.

We define the problem of inducing the intended program state from observations as follows.

Definition 2 (Intended Program Abduction). Given an abnormal component $c \in \Delta$ a model of a program $M(\Pi)$ and observations $M(I, O)$. We search for an alternative semantics $\hat{\Sigma}$ which transition the state of the program s_i into s_{i+1} following the operation of c, such that $M(\Pi) \cup \hat{\Sigma}_\Delta$ (c) $\vdash$ $M(I, O)$.

Applying the diagnosis to a single test case of the running example, focusing only on input and expected output, we gather a diagnosis $\Delta = \{5\}$. However, some of the justifications are not intuitive. For example, a naive solution with $\hat{\Sigma}(5)$ would replace the entire string in the last loop iteration, with the final output. While this solution is consistent, it is clearly not what the programmer intended.

The conceptual program $\hat{\Pi}$ is intangible, and a programmer's verbal descrip- tion of their mental model can be imprecise with implicit assumptions. Instead, we propose manual simulation as an interface between the program's conception and implementation, integrating it into the diagnostic model through the data D used. The conceptual program $\hat{\Pi}$ can execute as a mental (Forbus, 1990) or manual simulation, producing a trace of states that act as observations for the intermediate states of the actual implementation.

Running Example (Reversing Strings II). Consider a situation where a pro- grammar sketches out a trace of the program on a whiteboard by drawing the array (dominant data structure) and simulates its states s0 ... s3 step by step (see Fig. 2). On the whiteboard, the change in indexes for the two swapping positions are omitted, and managed they act as scaffolding for the implementation.

With intermediate observations from the manual simulation, the diagnosis $\Delta = \{5\}$ remains consistent, but the abduced semantics $\hat{\Sigma}(5)$ can only justify assignments by incorporating intermediate states s1 ... s3 (see Fig. 2). The remaining variables relevant to $M(\Pi)$, acting as scaffolding, are then subject to abduction and reconstruction by constraint programming.

This process, through abduced values, outlines what a potential program repair might be.

Conclusion

Manual simulation on paper, whiteboards or through conversation is essential for understanding or developing algorithms. It also generates intermediate observations that reflect the programmer's intention of the algorithm's behavior, making it an ideal source of information to diagnose programming mistakes.

We propose to use manual simulations, treating them as observations of computational states to inform diagnostic methods for debugging algorithm construction and implementation. These intermediate observations further constrain consistency-based diagnostic methods and help to derive potential fixes for problematic areas. In addition to a prototype implementation for diagnosing simple data structures, future work will include diagnosing bugs related to control flow and state.

Acknowledgments. The work presented in this paper has been carried out in context of the VoLL-KI project (grant 16DHKBI091), funded by Bundesministeriums für Bildung und Forschung (BMBF).

References

Abreu, R., Zoeteweij, P., Golsteijn, R., Van Gemund, A.J., 2009. A practical evaluation of spectrum-based fault localization. J. Syst. Softw. 82, 1780–1792.

Anderson, J.R., 1993. Rules of the mind. Psychology Press.

Bayerkuhnlein, M., Wolter, D., 2024. Model-Based Diagnosis with ASP for Non-groundable Domains, in: International Symposium on Foundations of Information and Knowledge Systems. Springer, pp. 363–380.

Bayerkuhnlein, M., Wolter, D., 2023. Model-Based-Diagnosis for Assistance in Programming Exercises, in: European Conference on Artificial Intelligence. Springer, pp. 459–470.

Chandra, K., Li, T.-M., Nigam, R., Tenenbaum, J., Ragan-Kelley, J., 2024. Watchat: Explaining perplexing programs by debugging mental models. ArXiv Prepr. ArXiv240305334.

Chmiel, R., Loui, M.C., 2004. Debugging: from novice to expert. ACM SIGCSE Bull. 36, 17–21.

Console, L., Friedrich, G., Dupré, D.T., 1993. Model-based diagnosis meets error diagnosis in logic programs, in: Automated and Algorithmic Debugging: First International

Workshop, AADEBUG'93 Linköping, Sweden, May 3–5, 1993 Proceedings 1. Springer, pp. 85–87.

Dovier, A., Formisano, A., Pontelli, E., 2005. A comparison of CLP(FD) and ASP solutions to NP-complete problems, in: Logic Programming: 21st International Conference, ICLP 2005, Sitges, Spain, October 2-5, 2005. Proceedings 21. Springer, pp. 67–82.

Forbus, K.D., 1990. The Qualitative Process Engine, in: Weld, D.S., de Kleer, J. (Eds.), Readings in Qualitative Reasoning About Physical Systems. Morgan Kaufmann, pp. 220–235. https://doi.org/10.1016/B978-1-4832-1447-4.50017-1

Johnson, W.L., Soloway, E., 1985. PROUST: Knowledge-based program understanding. IEEE Trans. Softw. Eng. SE-11, 267–275.

Johnson-Laird, P.N., 1989. Mental models, in: Posner, M.I. (Ed.), Foundations of Cognitive Science. The MIT Press, pp. 469–499.

Knobelsdorf, M., Romeike, R., 2008. Creativity as a pathway to computer science, in: Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education. pp. 286–290.

Magnani, L., 2011. Abduction, reason and science: Processes of discovery and explanation. Springer Science & Business Media.

Reiter, R., 1987. A theory of diagnosis from first principles. Artif. Intell. 32, 57–95.

Ross, B.J., 1997. Running programs backwards: the logical inversion of imperative computation. Form. Asp. Comput. 9, 331–348.

Shapiro, E.Y., 1982. Algorithmic program debugging. Yale University.

Sharmin, S., 2021. Creativity in CS1: a literature review. ACM Trans. Comput. Educ. TOCE 22, 1–26.

Weiser, M., 1984. Program slicing. IEEE Trans. Softw. Eng. SE-10, 352–357.

Wotawa, F., Dumitru, V.A., 2022. The Java2CSP debugging tool utilizing constraint solving and model-based diagnosis principles, in: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer, pp. 543–554.

Wotawa, F., Stumptner, M., Mayer, W., 2002. Model-Based Debugging or How to Diagnose Programs Automatically, in: Hendtlass, T., Ali, M. (Eds.), Developments in Applied Artificial Intelligence. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 746–757.

Zeller, A., Hildebrandt, R., 2002. Simplifying and isolating failure-inducing input. IEEE Trans. Softw. Eng. 28, 183–200.

Applying Ariadne: Practical Insights into Learning Style Identification via Hidden Markov Models

Flemming Bugert, Dominik Bittner, Timur Ezer, Vamsi Krishna Nadimpalli, Susanne Staufer, Lisa Grabinger, Florian Hauser, Jürgen Mottok

OTH Regensburg, Seybothstrasse 2,
93053 Regensburg, Germany
{firstname.lastname}@oth-regensburg.de

Abstract

With the use of learning management systems students benefit from being recommended suitable learning elements based on their individual needs. In doing so, recommendation algorithms are applied which first query the student's learning style. To improve the recommendation of learning elements a continuous analysis of the individual's learning style is required. A frequent questionnaire assessment would however be too time consuming. Instead, in a prior study an algorithm has been designed to identify changes in learning styles from the student's selection of learning elements. In this paper, we investigate the functionality of that algorithm by applying it on real student data. In particular, we test if the algorithm correctly indicates changes in learning styles. The utilised data is collected in our learning management system. To be precise, the data is obtained from 22 students enrolled in a software engineering course during the winter term of 2023/24. The data comprises two types of information for each student: 1) learning style collected at the start and end of the term, and 2) the user's actual selection of learning elements inside the learning management system. The uniqueness of this study lies in the data and the evaluation strategy based on it. Having the learning style at the end of the semester period as ground truth allows us to test if the algorithm operates correctly with actual user data from our learning management system. The results validate the behaviour of our algorithm, yet they strongly suggest the need for an adaptation. Further research is required on how to parameterise the underlying models.

Keywords Learning Styles, Hidden Markov Models, Empirical Evaluation

Introduction

Nowadays, learning management systems (LMSs) can often recommend suitable learning material to students for an improved learning success. In doing so, the underlying algorithms initially query the learning style. To continuously update the students' learning styles, the Ariadne algorithm has been designed [1]. It analyses the user's chosen learning elements inside a LMS and quantifies with a metric, the so-called support value, how much the user behaviour aligns with the initial questionnaire assessment. Therefore, the support value can be utilised to improve the recommendation of learning elements. The aim of this study is to validate the Ariadne algorithm based on real student data and thus provide practical insights. To do this, data was collected from students inside our LMS for a software engineering course from the winter term 2023/24. The data contains the learning style assessed at the start and end of the term. Also, information about the chosen learning elements inside the LMS is included. The learning style queried at the end of the semester period serves as ground truth for evaluation. This allows us to benchmark the current implementation and furthermore formulate directions for improvement. In summary, the present paper adds the following contributions:

- C1 Demonstrate and validate the Ariadne algorithm when practically applied,
- C2 Test and evaluate the Ariadne algorithm,
- C3 Specify needs for adaptation in the Ariadne algorithm.

The paper is structured as follows. First, theoretical foundations are introduced. Section 4 then explains the methodology of this study. After, the results are highlighted and discussed. Following that, section 6 presents the limitations of this study. Finally, section 7 draws a conclusion and shows directions for future work.

Related Work

Several techniques already exist to identify learning styles from user behaviour in LMSs [2]. García et al. for example apply Bayesian statistics to detect learning styles [3]. On the other side, Bernard et al. deploy deep learning techniques for a more precise learning style identification from user behaviour [2]. However, the proposed techniques are limited in their interpretability or require a large amount of data. More importantly, none of these strategies addresses the challenge of detecting changes in learning styles. Thus, the uniqueness of our approach becomes clear: this study evaluates an algorithm for continuous analysis of learning styles based on user behaviour. Additionally, the given data allows to provide understandable results.

Theoretical Background

The following part introduces the foundations of the present work. First, the utilised learning style theories and learning elements are described. After, we briefly present the LMS used to collect the data for this study. Then, the key features of the Ariadne algorithm are highlighted.

Felder and Silverman formulated a way how learners perceive and understand information by defining four categories. To keep the attention of the reader, this paper only covers the Active-Reflective category. Active learners prefer practicing through exercises, whereas reflective learners tend to think about the content first. [4–7]

Consequently, the theories imply that given the choice and sequencing of learning elements conclusions about the learning style can be drawn. The learning element types used for this study refer to the ones defined by Staufer et al. [8]. To gather the students' learning styles, the Index of Learning Styles (ILS) questionnaire is used in this study. In addition to the bare classification, it also maps the strength of each learning style to a value between 1 and 11. Thus, learners show a mild (1 and 3), balanced (5 and 7) or strong (9 and 11) coherence to their learning style. [9]

Managing Learning Content with Pythia

The LMS employed in our research project is called Pythia. It is a Moodle based software implementation. Among other things, it facilitates the integration of algorithms recommending a sequence of learning elements - the learning path. Notably, it allows us as well to collect user data and learning analytics effectively. [10]

Learning Style Identification with Ariadne

The Ariadne algorithm has been designed to counteract weaknesses of the ILS when used for curriculum design [7] by updating learning styles from the learners continuously based on their actual choice of learning elements. To do this, Hidden Markov Models (HMMs) are deployed, which model learning styles as hidden states and learning elements as observable objects. In the case of this work, the applied HMM has the hidden states active and reflective. Consequently, a chosen learning element can be analysed to deduce the corresponding most probable learning style. Like stated in Equation (1), the support value then is used to quantify how often the initial learning style is identified by the algorithm ($learning\ styles_{algorithm}$) in reference to the length of the learning path ($l_{learning\ path}$).

$$support\ value = \frac{learning\ styles_{algorithm}}{l_{learning\ path}} \tag{1}$$

This metric thus states how much the actual learning path aligns with the initial learning style assessment. The support value ranges from 0 (mismatch) to 1 (match). [1]

Methods

The following part first highlights the data used for this study. Then, the evaluation methodology is presented in brief.

Data

The data is obtained from 22 participants inside Pythia from a software engineering course of the winter semester period 2023/24. It contains results from the ILS rolled out in the start (Pre-Test) and the end (Post-Test) of the semester period as well as the students' learning paths inside Pythia. The data is made publicly available on Zenodo[2].

ILS Pre- and Post-Test

The results from Fig. 1 suggest the need for a continuous update of the learning style during the semester period. Four students change from Active to Reflective and seven students from Reflective to Active learning style.

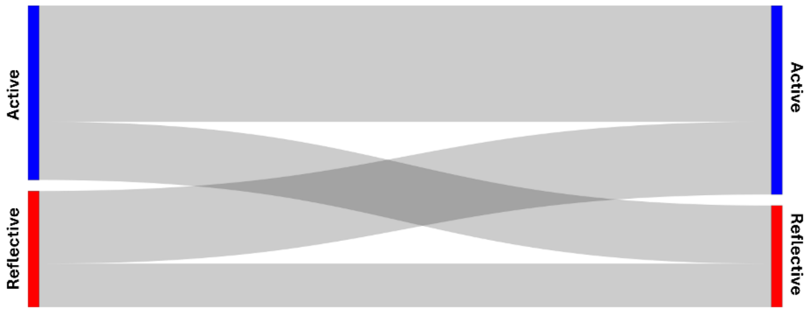


Fig. 1 Pre- (left) to Post-Test (right) Development of Learning Styles for the Active Reflective Learning Style Dimension

[2] 10.5281/zenodo.12594911

Moodle Learning Paths

The students' learning paths inside Pythia are chronological sequences of the learning elements with the Moodle status marked as done. After pressing the corresponding button, an event is triggered. This information is stored over the semester period and then used as data source for this study.

Evaluation Methodology

The evaluation methodology is displayed in Fig. 2. The Pre-Test learning style is used to initialise the HMM of the Ariadne algorithm. Then, the student's learning path serves as input for the algorithm to calculate the support value. Finally, we check if the outcome aligns with the Post-Test learning style. The results are obtained using the Python packages *hmm-learn 0.3.2* and *numpy 1.26.4*.

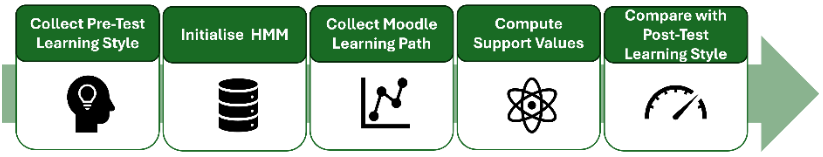

Collect Pre-Test Learning Style
Initialise HMM
Collect Moodle Learning Path
Compute Support Values
Compare with Post-Test Learning Style

Fig. 2 Workflow for the Evaluation of the Ariadne Algorithm

Results and Discussion

The results are listed in Table 1 and clustered based on Pre- and Post-Test outcomes. The questionnaire assessments show that mostly students with a mild learning style expression tend to change between Active and Reflective behaviour. On the contrary, students with a balanced or strong characteristic prefer to stay in their learning style. For these participants the support value mostly aligns with the questionnaire information. More importantly, the findings show that the algorithm is able to identify changes in the learning styles for specific students.

ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Change	–	–	–	–	–	–	–	–	↘	↘	↘	↘	–	–	–	↗	↗	↗	↗	↗	↗	↗
Pre-Test	3	9	9	7	5	5	3	5	3	1	5	5	3	9	1	5	1	1	1	1	1	1
Post-Test	5	7	9	7	5	5	5	3	1	1	1	3	3	9	3	1	1	1	3	5	3	3
SV	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.67	0.99	1.00	0.99	0.98	0.20	1.00	0.98	0.80	0.69	1.00	0.00	0.6

* The – symbol refers to a remain between Pre- and Post-Test learning style, whereas ↘ and ↗ indicate a change from Active to Reflective and Reflective to Active respectively

* Green entries indicate that the SV correctly suggests a change or remain of the learning style

Table 2 Support Values (SVs) from the algorithm for the learning paths from 22 students along with their ILS Pre- and Post-Test results.

For a better understanding, Fig. 3 illustrates the learning path for the student with ID 22. As stated in Table 1, the algorithm correctly doubts the Pre-Test learning style. Like shown in the learning path, the selection of quizzes leads to the algorithm suggesting that the student has Active learning tendencies as well. This is indicated by the support value of 0.60.

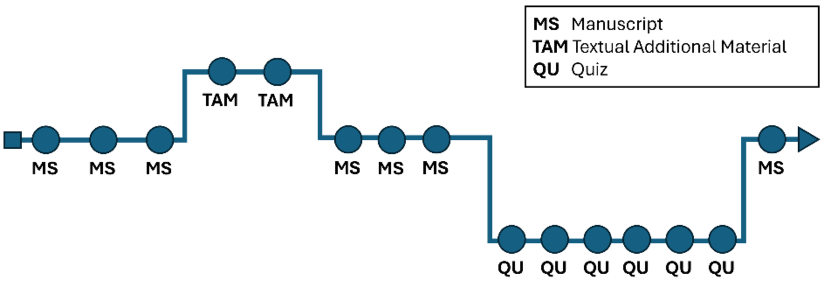


Fig. 3 Learning Path for ID 22

In summary, the results provide valuable insights into the behaviour of the algorithm. The findings imply that the Pre-Test learning style used to parameterise the HMM strongly influences the decision-making. We assume that the algorithm is overfitted to learners with a mild characteristic as correctly classifying these students is very important. Though, in this case the algorithm would not produce sensible outcomes for balanced and strong learners. More research is however required to confirm this statement.

Limitations

The major source of limitation to this study concerns the collected learning path data. Neither have the students been taught the right use of

Pythia, nor are we able to resolve whether the state *mark as done* has been applied correctly after the completion of a learning element. The users interact differently with our LMS. Hence, the learning path data for the Ariadne algorithm varies in input size and thus the findings are hard to compare. Also, we only analysed the data after completion of the term. However, the results can be different when running the algorithm during the semester period. Another limitation to the data is due to the fact that not all learning element types were equally present in the LMS. Hence, the data is biased to the given variety of learning elements. Finally, the questionnaire results are in general limited in their reliability. Students might have misunderstood questions or have not answered the questions consciously due to a lack of interest.

Conclusions and Future Work

In summary, this study practically evaluates the Ariadne algorithm and presents valuable insights into its behaviour when applied to real student data. The findings demonstrate that the Ariadne algorithm can be deployed to identify changes in the learning style and thus update learning path recommendation algorithms. Though, further adaptation is needed to refine the underlying computations. In particular, the results suggest that future work has to address how the learning style from the questionnaire affects the parameters of the HMM. Also, it is necessary to facilitate the extraction of more meaningful learning path data. Checking the usage time when accessing a learning element or analysing the user interaction with the learning element can provide more meaningful input data for the Ariadne algorithm.

Acknowlegdements

The present paper is supported by the 'German Federal Ministry of Education and Research' (BMBF) through the granting of the funding project HASKI (FKZ: 16DHBKI035).

References

[1] F. Bugert *et al.,* "Ariadne's Thread for Unravelling Learning Paths: Identifying Learning Styles via Hidden Markov Models," in *Ariadne's Thread for Unravelling Learning Paths: Identifying Learning Styles via Hidden Markov Models*, 2024, pp. 1–7.

[2] J. Bernard, T.-W. Chang, E. Popescu, and S. Graf, "Learning style Identifier: Improving the precision of learning style identification through computational intelligence algorithms," *Expert Systems with Applications*, vol. 75, pp. 94–108, 2017, doi: 10.1016/j.eswa.2017.01.021.

[3] P. García, A. Amandi, S. Schiaffino, and M. Campo, "Evaluating Bayesian networks' precision for detecting students' learning styles," *Computers & Education*, vol. 49, no. 3, pp. 794–808, 2007, doi: 10.1016/j.compedu.2005.11.017.

[4] R. M. Felder and B. A. Soloman, *Learning styles and strategies*.

[5] H. Fasihuddin, G. Skinner, and R. Athauda, "Towards an adaptive model to personalise open learning environments using learning styles," in *Proceedings of International Conference on Information, Communication Technology and System (ICTS) 2014*, Surabaya, Indonesia, 2014, pp. 183–188.

[6] P. Q. Dung and Adina Magda Florea, Eds., *An approach for detecting learning styles in learning management systems based on learners' behaviours*, 2012.

[7] R. M. Felder and J. Spurlin, "Applications, reliability and validity of the index of learning styles," *International journal of engineering education*, vol. 21, no. 1, pp. 103–112, 2005.

[8] S. Staufer, F. Hauser, L. Grabinger, D. Bittner, V. K. Nadimpalli, and J. Mottok, "LEARNING ELEMENTS IN ONLINE LEARNING MANAGEMENT SYSTEMS," in *ICERI2023 Proceedings*, Seville, Spain, 2023, pp. 3121–3130.

[9] B. A. Soloman and R. M. Felder, "Index of learning styles questionnaire," *NC State University. Available online at: http://www. engr. ncsu. edu/learningstyles/ilsweb. html (last visited on 14.05. 2010)*, vol. 70, 2005.

[10] S. Röhrl *et al.,* "PYTHIA - AI SUGGESTED INDIVIDUAL LEARNING PATHS FOR EVERY STUDENT," in *INTED2024 Proceedings*, Valencia, Spain, 2024, pp. 2871–2880.

For What Tasks and Purpose Do Computer Science Students Use Code Generators -- Preliminary Results of an Online-Study

Sonja Niemann[1] und Ute Schmid[1,2]

bidt – Bayerisches Forschungsinstitut
für Digitale Transformation, Munich, Germany
University of Bamberg, Germany

sonja.niemann@bidt.digital
ute.schmid@uni-bamberg.de

Abstract

Generative AI is applied in different educational contexts such as essay writing or translation of texts. A specific application of generative models is creation of program code. Educators in schools and universities face the challenge how to assure that students acquire relevant competencies while making use of generative AI tools. In this paper, we present preliminary results of an online study of the use of code generators by Computer Science (CS) students. Beginner students get to learn basic programming concepts and skills, advanced students get to learn how to solve complex programming tasks and create programs which are correct, efficient, modifiable, and well documented. With the presented online study we want to explore which tasks students solve with code generators and with what purpose. 285 students participated in the survey, new beginners as well as advanced students, with the goal to compare students who learned programming basics without code generators with those who always had access to them. Preliminary results show that students are very eager to learn coding skills and want to understand underlying concepts. Students who struggle with coding skills tend to ask code generators more often for explanations and use code generators for the purpose of understanding.

Students who chose not to use code generators are more skeptical of such systems and the mistakes they make. Our empirical results provide insights in how we can support students in acquiring coding skills while using code generators.

Keywords Code Generators, Programming Education, Programming Competencies

Introduction

Concepts and skills in writing programs are the core of computer science education and also a prerequisite for educational programs in artificial intelligence (AI). The current fast development of LLMs as code generators on the one hand provides a powerful tool to support students in many programming related tasks such as code generation from natural language specifications, explanation of code, test case generation, and code repair. On the other hand extensive use of code generators to solve programming assignments has the danger that students do not acquire relevant competencies and skills. That is, a core question for computer science and AI education is: Are students over-relying on code generators and missing critical programming skills? [1].

While first surveys try to gain insight in generative AI use at universities and their struggle with legal questions about copyrights and exams [6][2], there is little to no research about the usage habits of code generators among CS students. Therefore a small survey among CS students was conducted, asking about their code generator use when coding and their trust in such systems. Several aspects of code generator use are covered to understand the user behavior of this particular group. The item groups can be roughly described as followed: Frequency of use and systems used, specific tasks, purpose of use, trust in code generators, technical knowledge and personal goals. Questions about the tasks students give to code generators, allows us insight into the areas we can provide students with additional support.
First results suggest that first year students let code generators generate whole code blocks more often than master students. Master students use

code generators to correct their code more often and both groups use generative AI to explain things they don't understand. This paper will discuss preliminary results of the survey with a focus on the tasks students give to code generators and the purpose they are trying to achieve. The results of our survey will guide the next step of developing interfaces for code generators that are designed to cover the needs of CS students. Instead of condemning the use of code generators for CS students, the opportunity should be seized to harness its potential. To achieve this several options have come to our attention, for example providing explanations via preset prompts [4] or repairing buggy code and giving high precision feedback [7][5]. In an exploratory approach an interface combining the findings of the survey and sustainable LLM use will be developed.

Questionnaire and Sample Group

The Survey follows an exploratory approach since there is little to no comparable work. Other surveys concentrated on students in general, rather than focusing on a specific field of study. Instead, they provided an important overview [6, 2]. Previous work gives us the chance to use similar items and compare our specific sample group of CS Students to the more general sample groups.

Table 1: Item Groups; number of items

Group Title	Number of Items
1. Frequency and Systems	2-4
2. Tasks and Purpose	3
3. Perceived Advantage	1
4. Trust	19
5. Technical Knowledge	6
6. Personal Goals and Skills	4
7. Demographic Data	5

The items of the questionnaire are split into seven item groups that cover different topics. You can see the item groups with the number of items in

Table 1, the groups important for the results presented in this work will be explained closer. The first item group starts with participants selecting if they have heard of and used code generators like ChatGPT for example. Two of the possible answers suggest that they have not heard of or do not use code generators for coding task, those participants will skip
a few item groups after the first one. Item block two again is for participants who use code generators for coding, it is about the specific tasks they use code generators for as well as the purpose they hope to achieve. The first question of the item block asks about the setting they use code generators in, at work, at university, for private project, a combination is possible. The second item covers the purpose of the use, giving them three options to choose from, for example 'I use code generators to submit my programming tasks in time.'. The last question of the item block asks participants to rate how often they have specific tasks done by code generators. The answer format is a five point Likert scale, participants indicate the frequency code generators are used for each tasks individually. From item block four on all items are presented to every participant, no matter if they use code generators. As a first approach to the concept of trust in AI and trustworthy AI we used the 'Trust in Automation' questionnaire by Körber [3]. Item block six contains questions about their personal goals and skills, again giving them seven statements they have to agree or disagree to on a Likert scale, for example 'I really want to understand the content of my degree programme'. They also can rate to which extent they find other platforms like Stackoverflow or Youtube helpful. As a last item in this group they can choose to give additional information in a text field. The complete survey can be found in the appendix.

The survey was distributed via several channels, with the goal to reach bachelor and master students in computer science or closely related fields. Several professors, a student council and the 'Junge Gesellschaft für Informatik' were contacted and asked to distribute the survey to the students via email. Students were free to participate. Over four weeks data from 342 participants were collected, 289 fulfilled the criteria of being a CS student or from a closely related fields as well as completing the survey. The group will be split to compare students who have started university before popular LLMs like ChatGPT have been announced and students who have been exposed to LLMs since day one of their degree. Everyone with more than 4 semester in a bachelor degree will be considered a higher level CS student. The two groups formed will be called 'beginners' and 'advanced' and have roughly the same amount of participants with n = 144 beginners and n = 145 advanced. The beginner group has an average age of 22 with a average of 3 semesters studied. The advanced group has an average age of 26 and an average of 7 semester studied. In a next step students who do not use code generators were separated from both groups, in total 38 students formed the group of non users and leaves the original two groups with n = 127 advanced students and n = 124 beginners.

Results

The survey covers several variables that are expect to influence each other, coding skills, trust, knowledge about code generator to name a few. Aspects we want to take a closer look at are students mindset towards coding and perceived struggles and goals they personally have. Some of the answers students have written as additional information allow us a more detailed, but subjective look at their experiences.

One rather negative possibility is that students do not see the need in understanding and learning of coding skills anymore and therefore use code generators to finish their assignment in time and get good grades.

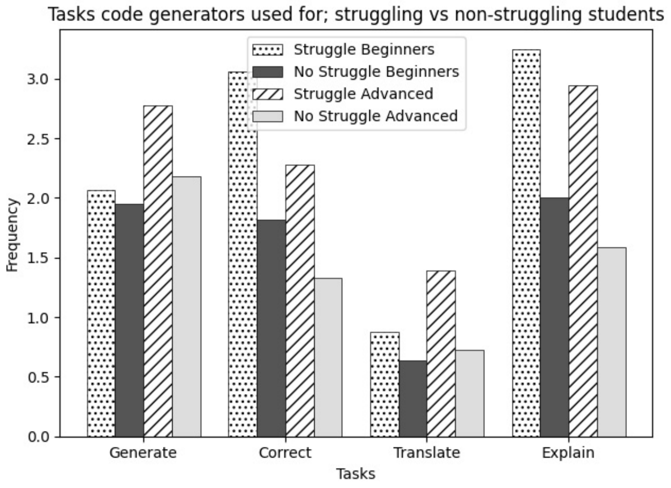


Figure 1 Tasks different student groups use code generators for; Group sizes: SB = 16, NSB = 77, SA = 18, NSA = 84

Out of all 251 students using code generators only ten declared that they do not believe acquiring coding skills is important for them. The most important aspect is understanding the content of their study program, with a mean of 4.47 on a 5 point Likert scale (1 meaning not important at all, 5 meaning very important) over all students using code generators. This is closely followed by a mean of 4,29 for the item 'Learning programming skills is important to me'. Further participants seem to enjoy programming and most of them have a feeling of self-efficacy when asked if they are capable of solving coding tasks on their own. There is a smaller group with n = 34 that is less confident in their programming abilities compared to their fellow students, n = 57 would neither agree nor disagree to the statement and n = 160 felt confident that they were as good as their class mates. We found a difference in the use of code generators in the two extreme groups, the 57 indecisive students are excluded in the following results. As displayed in Figure 1, 195 students are divided into four groups depending on their self-reported coding skills: advanced students(n = 102) and beginners(n = 93) and each of them are subdivided again into students who report struggles with coding tasks (advanced n = 18, beginners n = 16) and those who are confident in their abilities(advanced n = 84, beginners n = 77). The X-axis show the different tasks students used the code generators for and the y-axis displays the average reported frequency the groups would use code generators for the specific task. Using code generators to explain code is more important to students who struggle with coding tasks, no matter if advanced or beginner. Generating code is more often used by the advanced group that struggle with coding. While beginners who struggle rely on code generators to correct their code. In a second step the purpose these groups want to find in code generators are analyzed. Figure 2 displays the purpose students hope to find in code generators on the x-axis, with the groups and y-axis staying the same as in the first graphic. Understanding tasks is most important to struggling students coinciding with the findings in Figure 1, where explaining was the most used tasks. They also want to learn to code but more often rely on code generators to finish assignments in time than students who are confident in their programming skills.

Participants were free to give additional information about their experience with code generators, or why they chose not to use them.

First some insights into the answers from students who use code generators: Common ground for the fast majority is that code generators can solve easy task or smaller chunks for a bigger task, but are perceived as less helpful with complex tasks or very specific questions. One comment sums up a lot of their experiences 'Copilot is useful to make automated repetitive or standard code blocks, e.g. creating for-loops to alter arrays. For specific or complex tasks it regularly fails.' Some even went further and described code generators as 'sparing partner' or 'pair programmer' that could give ideas how something can be done or one could discuss new approaches with. One last example how students developed their very own approaches to code generators is a person who reported his code documentation got a lot better and more detailed, because he is making sure to give needed details to his LLM of choice.

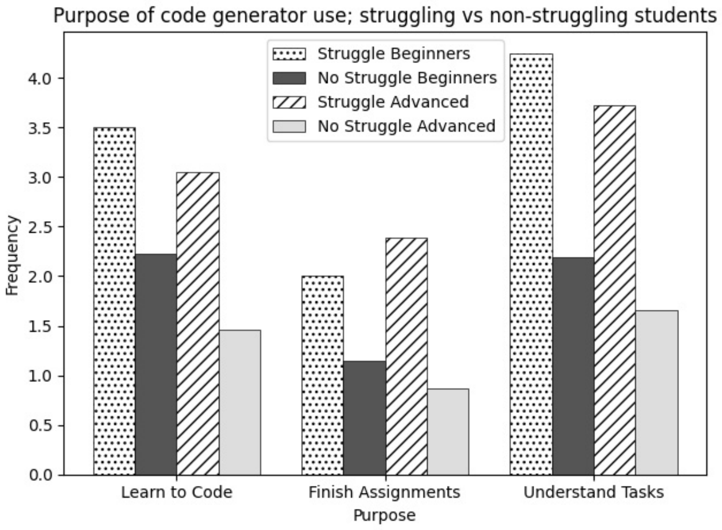


Figure 2 Purpose of code generator use between different student groups; Group sizes: SB = 16, NSB = 77, SA = 18, NSA = 84

On the other hand we have CS students who decided not to use code generators, they agree on certain aspects as well. One they do not trust AI to be correct and are afraid to not find the mistakes AI makes. Two they want to understand the concepts and code they create and believe they wouldn't if they used code generators. And three some are concerned about their academic integrity 'I don't trust them to generate the best solution and I won't learn anything from just copying their solution anyways. I also would rather not get in trouble for academic misconduct.' Some expressed concerns about their privacy and data use in models like ChatGPT and said they don't trust the people behind the systems.

Discussion

It has to be pointed out that this survey is not representative, a known problem is a selective bias in such surveys. Students who choose to participate in such surveys are likely to be motivated and interested in their study, students who truly struggle are less likely to participate. However, the results can still be used to listen to students' needs and guide future research projects. On the positive site it was shown that the students who participated are eager to learn and understand programming concepts, despite the current discussion that they might rely on generative AI too much. The user patterns, different types of students show, can help us to develop support structures to ensure their needs are met. For example, students who are struggling with their programming skills seek to understand the concepts. In a next phase Interfaces can be designed to explore what type of support gets students the needed information, for example through pre-set prompts [4]. Students would not have to figure out what to ask and how to write a good prompt and could focus on understanding. This group also tends to rely on code generators to hand in assignments in time, therefore are in danger to fall behind with understanding the concepts. Another group that could profit from input are students that up until now do not use code generators. Research should look into trustworthiness of LLMs to enable students to use newest technology without fear. The trust questionnaire in this survey might bring some more insights into this topic.

References

[1] Becker, B.A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., Santos, E.A.: Programming is hard - or at least it used to be. In: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1. pp. 500–506. ACM, New York, NY, USA (2023).

[2] Hüsch, M.: Künstliche Intelligenz im Studium: Nicht überall auf Kurs (2024),https://www.che.de/2024/kuenstliche-intelligenz-im-studium-noch-nicht-in-allen-faechern-auf-kurs/

[3] Körber, M.: Theoretical considerations and development of a questionnaire to measure trust in automation. In: Bagnara, S., Tartaglia, R., Albolino, S., Alexander, T., Fujita, Y. (eds.) Proceedings of the 20th Congress of the International Ergonomics Association (IEA 2018). pp. 13–30. Springer, Cham (2019)

[4] Nam, D., Macvean, A., Hellendoorn, V., Vasilescu, B., Myers, B.: Using an LLM to help with code understanding. In: Roychoudhury, A., Paiva, A., Abreu, R., Storey, M. (eds.) Proceedings of the 46th IEEE/ACM International Conference on Software Engineering. pp. 1–13. ACM Digital Library (2024).

[5] Phung, T., Cambronero, J., Gulwani, S., Kohn, T., Majumdar, R., Singla, A., Soares, G.: Generating high-precision feedback for programming syntax errors using large language models.

[6] Schlude, A., Mendel, U., Stürz, R.A., Fischer, M.: Verbreitung und Akzeptanz generativer KI an Schulen und Hochschulen (2024), bidt, Munich.

Appendix

Summary of the Items of the Questionnaire

Item Block 1: Frequency of use and system selection

1.1 Have you heard of code generators for creating and/or checking programme code and have you ever used systems such as ChatGPT or Copilot for this purpose?

Answer options:

- *Yes, I have heard of it and use it regularly for these purposes.Yes*
- *I have heard of it and use it sometimes for these purposes.*
- *Yes, I have heard of it and used it once or twice for these purposes.*
- *Yes, I have heard of it, but never used it for these purposes.*
- *No, I had never heard of it before this survey.*

1.2 Indicate to what extent you agree with the following statement about yourself: I know and understand the principles of generative AI.

Answer options: 5 point Likert scale 'Strongly disagree' to 'Strongly agree'

1.3 I use the following code generators for programming/correcting code or related tasks:
Answer options:

- *OpenAI GPT-3.5*
- *OpenAI GPT-4*
- *GitHub Copilot*
- *Amazon Codewhisperer*
- *Mistral AI*
- *Others*

1.4 What other code generators do you use?
Answer in open text field

Item Block 2: Tasks and Purposes

2.1 When do you use code generators for programming tasks
Answer options (multiple choice):

- *At University*
- *private projects*
- *at work*
- *others*

2.2 With what intention do you use code generators when using them for programming tasks?
Answer Options (5 point Likert scale for each statement from 'Never' to 'Always'):

- *I use code generators to learn to programme better.*
- *I use code generators to submit my programming tasks in time.*
- *I use code generators to understand programming tasks properly.*

2.3 How often do you have code generators perform the following specific tasks?

Answer Options (5 point Likert scale for each statement from 'Never' to 'Always'):

- *Generate program code*
- *check program code*
- *translate program code into other language*
- *explain program code*

Item Block 3: Advantages

3.1 Code generators have helped me...

Answer Options(5 point Likert scale for each statement from 'Strongly disagree' to 'Strongly agree'):

- *...to improve my performance.*
- *...to improve my programming skills.*
- *...to improve my knowledge about programming.*
- *...to improve my knowledge about programming concepts.*
- *...to save time when coding*
- *...to have more fun with programming.*

Item Block 4: Trust

19 questions from the Trust in Automation Questionnaire, see Körber (2019)

Item Block 5: Technical Knowledge

Read through the statements about code generators and indicate whether they are correct or incorrect.

Answer Options(for each statement 'Correct', 'Incorrect' and 'I don't know'):

- *'Code generators such as ChatGPT and the similar systems are large language models with transformer architecture.'*
- *'Code generators search the Internet for the right answers.'*
- *'The main function of the "self-attention" mechanism is the modelling of the time sequence of data points.'*
- *'The transformer architecture can be divided into encoder and decoder phases.'*
- *'Code generators can solve every task equally well.'*
- *'Code generators can provide incorrect answers.'*

Item Block 6: Personal Motivation and Preferred Platforms

6.1 How would you rate the usefulness of the following platforms for solving programming tasks?
Answer Options(5 point Likert scale from 'Never useful' to 'Always useful' and 'I don't use it'):

- *Stackoverflow*
- *Reddit*
- *StackExchange*
- *GeeeksforGeeks*
- *Youtube*

6.2 Read the following statements and indicate to what extent they apply to you personally.
Answer Options(5 point Likert scale from 'Strongly disagree' to 'Strongly agree'):

- *'I really want to understand the content of my degree program'*
- *'I want good grades', 'I enjoy programming'*
- *'Learning to programme is more difficult for me than for my fellow students.'*
- *'I believe that I can solve the programming tasks in my course independently.'*
- *'I regularly attend tutorials/practice sessions.'*
- *'Learning to programme is important to me.'.*

6.3 What works particularly well when you use code generators for programming, and what doesn't? Is there anything else you would like to share? OR Why don't you use code generators? Is there anything else you would like to share?
Answer Options: Open text field

Item Block 7: Demographics

7.1 Please enter your age in years.
7.2 Which gender do you identify with? (male/female/divers/no answer)
7.3 Which degree program are you enrolled in? Please indicate whether you are enrolled in a Bachelor's (BSc) or Master's (MSc) program.

7.4 What semester are you studying in?

7.5 Which university are you enrolled at?

University
of Bamberg
Press

Artificial Intelligence (AI) methods for education have been a topic of AI research since many decades with focus on Intelligent Tutoring Systems (ITS). ITS typically address a specific knowledge domain with a focus on individual learning support based on learning by problem solving, identification of misconceptions, and tailored feedback. Methods often are a combination of knowledge-based and machine learning approaches. Since the emergence of technologies based on Large Language Models (LLMs), the use of LLMs in educational contexts has received much attention, resulting in novel approaches.

The Second Workshop on Artificial Intelligence for Artificial Intelligence Education (AI4AILearning) has been held on September 24, 2024, in conjunction with the 47th German Conference on Artificial Intelligence in Würzburg, Germany. The workshop has a focus on research of AI methods for teaching AI competencies including programming skills. It provides a platform for exchange of ideas and experiences under the general theme of AI for education, specialising on university education in AI. The workshop includes interdisciplinary contributions from cognitive science and education technology.

ISBN: 978-3-98989-054-1

9 783989 890541

www.uni-bamberg.de/ubp/